Vidya Jyothi Institute of Technology

(An Autonomous Institution)

(Accredited by NAAC, Approved by AICTE & Permanently Affiliated to JNTUH)

DEPARTMENT OF INFORMATION TECHNOLOGY

Course Name : Computer Networks & Operating Systems Lab

Course ID : A35588

Prerequisites : C Programming

Name of the Faculty: Mrs.D.Anuradha/D.Sravanthi

B.Tech-Semseter

(R19)

Program Outcomes-POs:

1. **Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes:

Program Specific Outcomes	Statements
PSO1	Enhanced ability in applying mathematical abstraction and algorithmic design along with programming tools to solve complexity involved in efficient programming.
PSO2	Develop effective software skills and documentation ability for graduates to become Employable/ Higher studies/Entrepreneur researcher.

Course name: COMPUTER NETWORKS & OPERATING SYSTEMS LAB

After completing this course the student must demonstrate the knowledge and ability to	
CO1	Implement various CPU scheduling algorithms
CO2	Apply the memory management techniques
CO3	Write Programs on File allocation strategies
CO4	Implement various algorithms for error detection and correction
CO5	Implement Algorithms on Shortest path routing
CO6	Write a program for congestion control

Course name: Computer Networks & Operating Systems Lab

	PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**
CO 1	3	3	3	3	3	3	2	2	2	1	2	2
CO 2	3	3	3	3	3	3	2	2	2	1	2	2
CO 3	3	3	3	3	2	2	2	2	2	1	2	2
CO 4	3	3	3	3	3	3	2	2	2	1	2	2
CO 5	3	3	3	3	2	2	2	2	2	1	2	2
CO 6	3	3	3	3	3	3	2	2	2	1	2	2
Avg	**3**	**3**	**3**	**3**	**3**	**3**	**2**	**2**	**2**	**1**	**2**	**2**

Course name: Computer Networks and Operating Systems Lab

	PSO1	PSO2
CO1	3	3
CO2	3	3
CO3	3	3
CO4	3	3
CO5	3	3
AVG	3	3

INDEX

OS LAB PROGRAMS

INDEX
CN LAB PROGRAMS

Week: 1

Basic commands in Linux

1.1 File handling utilities

A) Cat

Command:

cat linux command concatenates files and print it on the standard output.

SYNTAX:

The Syntax is

cat [OPTIONS] [FILE]...

OPTIONS:

-A Show all.

-b Omits line numbers for blank space in the output.

-e A $ character will be printed at the end of each line prior to a new line. -E Displays a $ (dollar sign) at the end of each line.

-n Line numbers for all the output lines.

-s If the output has multiple empty lines it replaces it with one empty line. -T Displays the tab characters in the output.

Non-printing characters (with the exception of tabs, new-lines and form-feeds) -v are printed visibly.

Example:

To Create a new file: cat

> file1.txt

This command creates a new file file1.txt. After typing into the file press control+d (^d) simultaneously to end the file.

1. **To Append data into the file:**
   cat >> file1.txt
   To append data into the same file use append operator >> to write into the file, else the file will be overwritten (i.e., all of its contents will be erased).

2. **To display a file:**
   cat file1.txt
   This command displays the data in the file.

3. **To concatenate several files and display:**
   cat file1.txt file2.txt

   The above cat command will concatenate the two files (file1.txt and file2.txt) and it will display the output in the screen. Some times the output may not fit the monitor screen. In such situation you can print those files in a new file or display the file using less command.

   cat file1.txt file2.txt | less

4. **To concatenate several files and to transfer the output to another file.**
   cat file1.txt file2.txt > file3.txt
   In the above example the output is redirected to new file file3.txt. The cat command will create new file file3.txt and store the concatenated output into file3.txt.

B) mv COMMAND:

mv command which is short for move. It is used to move/rename file from one directory to another. mv command is different from cp command as it completely removes the file from the source and moves to the directory specified, where cp command just copies the content from one file to another.

SYNTAX:

The Syntax is

mv [-f] [-i] oldname newname

OPTIONS:

This will not prompt before overwriting (equivalent to --reply=yes).

mv -f will

-f move the file(s) without prompting even if it is writing over an existing target.

-i Prompts before overwriting another file.

EXAMPLE:

1. **To Rename / Move a file:**
   mv file1.txt file2.txt
   This command renames file1.txt as file2.txt

2. **To move a directory**
   mv hscripts tmp

In the above line mv command moves all the files, directories and sub-directories from hscripts folder/directory to tmp directory if the tmp directory already exists. If there is no tmp directory it rename's the hscripts directory as tmp directory.

3. **To Move multiple files/More files into another directory**
   mv file1.txt tmp/file2.txt newdir
   This command moves the files file1.txt from the current directory and file2.txt from the tmp folder/directory to newdir.

c) rm COMMAND:

rm linux command is used to remove/delete the file from the directory.

SYNTAX:

The Syntax is

rm [options..] [file | directory]

OPTIONS:

-f	**Remove all files in a directory without prompting the user.**
	Interactive.
	With this option, rm prompts for confirmation before removing any files
-i	**Recursively remove directories and subdirectories in the argument list**
-r (or) –R	**The user is normally prompted for removal of any write protected files which the directory contains**

EXAMPLE:

1. **To Remove / Delete a**
   file: rm file1.txt
   Here rm command will remove/delete the file file1.txt.

2. **To delete a directory**
   tree: rm -ir tmp
   This rm command recursively removes the contents of all subdirectories of the tmp directory, prompting you regarding the removal of each file, and then removes the tmp directory itself.

3. **To remove more files at**
   once rm file1.txt file2.txt
   rm command removes file1.txt and file2.txt files at the same time.

d) cp COMMAND:

cp command copy files from one location to another. If the destination is an existing file, then the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten).

SYNTAX:

The Syntax is

cp [OPTIONS]... SOURCE DEST

cp [OPTIONS]... SOURCE... DIRECTORY

cp [OPTIONS]... --target-directory=DIRECTORY SOURCE...

OPTIONS:

-a	**same as –dpR**
-backup[=CONTROL]	**make a backup of each existing destination file**
-b	**like –backup but does not accept argument.**
-f	**if an existing destination file cannot be opened, remove it and try again**
-p	**same as – preserve= mode, ownership, timestamps.**

ii) Directory commands

a) mkdir COMMAND: mkdir command is used to create one or more directories.

SYNTAX:

The Syntax is

mkdir [options] directories

OPTIONS:

-m	**Set the access mode for the new directories.**
-p	**Create intervening parent directories if they don't exist.**
-v	**Print help message for each directory created.**

EXAMPLE:

1. **Create directory:**

mkdir test

The above command is used to create the directory 'test'.

2. **Create directory and set permissions:**
   mkdir -m 666 test
   The above command is used to create the directory 'test' and set the read and write permission.

b) cd COMMAND:

cd command is used to change the directory.

SYNTAX:

The Syntax is

cd [directory | ~ | ./ | ../ | -]

OPTIONS:

-L Use the physical directory structure.

-P Forces symbolic links.

EXAMPLE:

1. cd linux-command

This command will take you to the sub-directory(linux-command) from its parent directory.

c) ls COMMAND

ls without any additional parameters, the program will list the contents of the current directory in short form.

ls [option(s)] [file(s)]

-l detailed list

-a displays hidden files

d) rmdir COMMAND:

rmdir command is used to delete/remove a directory and its subdirectories.

SYNTAX:

The Syntax is

rmdir [options..] Directory

OPTIONS:

-p	**Allow users to remove the directory dirname and its parent directories which becomes empty**

EXAMPLE:

1. **To delete/remove a directory**
   rmdir tmp
   rmdir command will remove/delete the directory tmp if the directory is empty.
2. **To delete a directory tree: rm -**
   ir tmp
   3. This command recursively removes the contents of all subdirectories of the tmp directory, prompting you regarding the removal of each file, and then removes the tmp directory itself.

Week2: Simulate the following CPU Scheduling Algorithms using C.

a) FCFS b) SJF

a) Aim: Write a C program to implement the various process scheduling mechanisms such

Algorithm for FCFS scheduling:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Set the waiting of the first process as '0' and its burst time as its turn around time
Step 5: for each process in the Ready Q calculate
- (a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
- (b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate
- (a) Average waiting time = Total waiting Time / Number of process
- (b) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

/* FCFS SCHEDULING ALGORITHM */

```
#include<stdio.h>
void main()
{
 int i,n,sum,wt,tat,twt,ttat;
 int t[10];
 float awt,atat;
 clrscr();
 printf("Enter number of processors:\n");
 scanf("%d",&n);
 for(i=0;i<n;i++)
{
          printf("\n Enter the Burst Time of the process %d",i+1);
          scanf("\n %d",&t[i]);
 }
 printf("\n\n FIRST COME FIRST SERVE SCHEDULING ALGORITHM \n");
 printf("\n Process ID \t Waiting Time \t Turn Around Time \n");
 printf("1 \t\t 0 \t\t %d \n",t[0]);
 sum=0;
 twt=0;
 ttat=t[0];
 for(i=1;i<n;i++)
{
          sum+=t[i-1];
          wt=sum;
          tat=sum+t[i];
          twt=twt+wt;
          ttat=ttat+tat;
          printf("\n %d \t\t %d \t\t %d",i+1,wt,tat);
          printf("\n\n");
 }
  awt=(float)twt/n;
  atat=(float)ttat/n;
  printf("\n Average Waiting Time %4.2f",awt);
  printf("\n Average Turnaround Time %4.2f",atat);
  getch();
}
```

OUTPUT:

Enter number of processors:3

Enter the Burst Time of the process 1: 2

Enter the Burst Time of the process 2: 5

Enter the Burst Time of the process 3: 4

FIRST COME FIRST SERVE SCHEDULING ALGORITHM

Process ID	Waiting Time	Turn Around Time
1	0	2
2	2	7
3	7	11

Average Waiting Time 3.00
Average Turnaround Time 6.67

b) Aim: Write a C program to implement the various process scheduling mechanisms such as SJF Scheduling .

Algorithm for SJF

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.
Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.
Step 6: For each process in the ready queue, calculate
- (c) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
- (d) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate
- (c) Average waiting time = Total waiting Time / Number of process
- (d) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

/* SJF SCHEDULING ALGORITHM */

```
#include<stdio.h>
void main()
{
 int i,j,k,n,sum,wt[10],tt[10],twt,ttat;
 int t[10],p[10];
 float awt,atat;
 clrscr();
 printf("Enter number of process\n");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
        printf("\n Enter the Burst Time of Process %d",i);
        scanf("\n %d",&t[i]);
 }
 for(i=0;i<n;i++)
 p[i]=i;
 for(i=0;i<n;i++)
{
        for(k=i+1;k<n;k++)
        {
                if(t[i]>t[k])
                {
                        int temp;
                        temp=t[i];
                        t[i]=t[k];
                        t[k]=temp;
                        temp=p[i];
                        p[i]=p[k];
                        p[k]=temp;
                }
            }
            printf("\n\n SHORTEST JOB FIRST SCHEDULING ALGORITHM");
            printf("\n PROCESS ID \t BURST TIME \t WAITING TIME \t
                        TURNAROUND TIME \n\n");
            wt[0]=0;
            for(i=0;i<n;i++)
            {
                    sum=0;
              for(k=0;k<i;k++)
                    {
                            wt[i]=sum+t[k];
```

```
                        sum=wt[i];
                }
        }
        for(i=0;i<n;i++)
        {
                tt[i]=t[i]+wt[i];
        }
        for(i=0;i<n;i++)
        {
                printf("%5d \t\t5%d \t\t %5d \t\t %5d \n\n",p[i],t[i],wt[i],tt[i]);
        }
        twt=0;
        ttat=t[0];
        for(i=1;i<n;i++)
        {
            twt=twt+wt[i];
            ttat=ttat+tt[i];
        }
        awt=(float)twt/n;
        atat=(float)ttat/n;
        printf("\n AVERAGE WAITING TIME %4.2f",awt);
        printf("\n AVERAGE TURN AROUND TIME %4.2f",atat);
        getch();
  }
}
```

OUTPUT:

```
Enter number of process 3
 Enter the Burst Time of Process 04
 Enter the Burst Time of Process 13
 Enter the Burst Time of Process 25
 SHORTEST JOB FIRST SCHEDULING ALGORITHM
 PROCESS ID     BURST TIME     WAITING TIME    TURNAROUND TIME

   1        3              0          3

   0        4              3          7

   2        5              7          12

 AVERAGE WAITING TIME 3.33
 AVERAGE TURN AROUND TIME 7.33
```

Week 3: Simulate the following CPU Scheduling Algorithms using C.

a) Priority b) Round Robin

a) Aim: Write a C program to implement the various process scheduling mechanisms such as Priority Scheduling.

Algorithm for Priority Scheduling:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Sort the ready queue according to the priority number.
Step 5: Set the waiting of the first process as '0' and its burst time as its turn around time
Step 6: For each process in the Ready Q calculate
- (e) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
- (f) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 7: Calculate
- (e) Average waiting time = Total waiting Time / Number of process
- (f) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

/* PRIORITY SCHEDULING */

```
#include  <stdio.h>
#include <conio.h>
void main()
{
        int i,j,n,tat[10],wt[10],bt[10],pid[10],pr[10],t,twt=0,ttat=0;
        float awt,atat;
        printf("\n----------PRIORITY SCHEDULING ----------\n");
        printf("Enter the No of Process: ");
        scanf("%d", &n);
        for (i=0;i<n;i++)
                {
                        pid[i] = i;
                        printf("Enter the Burst time of Pid %d : ",i);
                        scanf("%d",&bt[i]);
                        printf("Enter the Priority of Pid %d : ",i);
                        scanf ("%d",&pr[i]);
                }
// Sorting start
                for (i=0;i<n;i++)
                        for(j=i+1;j<n;j++)
                        {
if (pr[i] > pr[j] )
{
t = pr[i];
pr[i] = pr[j];
pr[j] = t;
t = bt[i];
bt[i] = bt[j];
bt[j] = t;

t = pid[i];
pid[i] = pid[j];
pid[j] = t;
}
}

// Sorting finished

        tat[0] = bt[0];
        wt[0] = 0;
```

```
for (i=1;i<n;i++)
        {
        wt[i] = wt[i-1] + bt[i-1];
        tat[i] = wt[i] + bt[i];
        }
printf("\n.....\n");
printf("Pid\t Priority\tBurst time\t WaitingTime\tTurnArroundTime\n");
printf("\n.....\n");
        for(i=0;i<n;i++)
        {
                printf("\n%d\t\t%d\t%d\t\t%d\t\t%d",pid[i],pr[i],bt[i],wt[i],tat[i]);
        }
for(i=0;i<n;i++)
        {
                ttat = ttat+tat[i];
                twt = twt + wt[i];
        }
awt = (float)twt / n;
atat = (float)ttat / n;
printf("\n\nAvg.Waiting Time: %f\nAvg.Turn Around Time: %f\n",awt,atat);
}
```

OUTPUT:

```
----------PRIORITY SCHEDULING----------
Enter the No of Process:        4
Enter the Burst time of Pid 0 : 2
Enter the Priority   of Pid 0 :   3
Enter the Burst time of Pid 1 : 6
Enter the Priority   of Pid 1 :   2
Enter the Burst time of Pid 2 : 4
Enter the Priority   of Pid 2 :   1
Enter the Burst time of Pid 3 : 5
Enter the Priority   of Pid 3 :   7
-------------------------------------------------------
Pid     Priority      Burst time      WaitingTime    TurnArroundTime

-------------------------------------------------------

2           1           4               0              4
1           2           6               4              10
0           3           2               10             12
3           7           5               12             17

Avg.Waiting Time: 6.500000
Avg.Turn Around Time: 10.750000
```

b) Aim: Write a C program to implement the various process scheduling mechanisms such as Round Robin Scheduling.

Algorithm for RR

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Calculate the no. of time slices for each process where
No. of time slice for process(n) = burst time process(n)/time slice
Step 5: If the burst time is less than the time slice then the no. of time slices =1.
Step 6: Consider the ready queue is a circular Q, calculate
(a) Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)
(b) Turn around time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).
Step 7: Calculate
(g) Average waiting time = Total waiting Time / Number of process
(h) Average Turnaround time = Total Turnaround Time / Number of process
Step 8: Stop the process

/* ROUND ROBIN SCHEDULING ALGORITHM */

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int ts,pid[10],need[10],wt[10],tat[10],i,j,n,n1;
        int bt[10],flag[10],ttat=0,twt=0;
         float awt,atat;
         clrscr();
         printf("\t\t ROUND ROBIN SCHEDULING \n");
         printf("Enter the number of Processors \n");
         scanf("%d",&n);
         n1=n;
         printf("\n Enter the Timeslice \n");
         scanf("%d",&ts);
         for(i=1;i<=n;i++)
         {
                printf("\n Enter the process ID %d",i);
                scanf("%d",&pid[i]);
                printf("\n Enter the Burst Time for the process");
                scanf("%d",&bt[i]);
                 need[i]=bt[i];
        }
        for(i=1;i<=n;i++)
        {
                flag[i]=1;
                 wt[i]=0;
        }
         while(n!=0)
        {
                for(i=1;i<=n;i++)
                {
                        if(need[i]>=ts)
                            {
                                for(j=1;j<=n;j++)
                                {
                                        if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
                                        wt[j]+=ts;
                                }
                                 need[i]-=ts;
                                 if(need[i]==0)
                                 {
```

```
                                        flag[i]=0;
                                        n--;
                                }
                        }
                        else
                        {
                                for(j=1;j<=n;j++)
                                {
                                        if((i!=j)&&(flag[i]==1)&&(need[j]!=0))
                                        wt[j]+=need[i];
                                }
                                need[i]=0;
                                n--;
                                flag[i]=0;
                        }
                }
        }
        for(i=1;i<=n1;i++)
        {
                tat[i]=wt[i]+bt[i];
                twt=twt+wt[i];
                ttat=ttat+tat[i];
        }
        awt=(float)twt/n1;
        atat=(float)ttat/n1;
        printf("\n\n ROUND ROBIN SCHEDULING ALGORITHM \n\n");
        printf("\n\n Process \t Process ID \t BurstTime \t Waiting Time \t TurnaroundTime \n ");
        for(i=1;i<=n1;i++)
        {
         printf("\n %5d \t %5d \t\t %5d \t\t %5d \t\t %5d \n", i,pid[i],bt[i],wt[i],tat[i]);
         }
        printf("\n The average Waiting Time=4.2f",awt);
        printf("\n The average Turn around Time=4.2f",atat);
        getch();
}
```

OUTPUT:

ROUND ROBIN SCHEDULING

Enter the number of Processors 4

Enter the Timeslice 5

Enter the process ID 1 5

Enter the Burst Time for the process 10

Enter the process ID 2 6

Enter the Burst Time for the process 15

Enter the process ID 3 7

Enter the Burst Time for the process 20

Enter the process ID 4

Enter the Burst Time for the process 25

ROUND ROBIN SCHEDULING ALGORITHM

Process	Process ID	BurstTime	Waiting Time	TurnaroundTime
1	5	10	15	25
2	6	15	25	40
3	7	20	25	45
4	8	25	20	45

The average Waiting Time=4.2f

The average Turn around Time=4.2f

Week 4: Simulate Paging Technique of Memory Management using C
Aim: To implement the Memory management policy- Paging.

Algorithm:

Step 1: Read all the necessary input from the keyboard.

Step 2: Pages - Logical memory is broken into fixed - sized blocks.

Step 3: Frames – Physical memory is broken into fixed – sized blocks.

Step 4: Calculate the physical address using the following

Physical address = (Frame number * Frame size) + offset

Step 5: Display the physical address.

Step 6: Stop the process.

/* Memory Allocation with Paging Technique */

```
#include  <stdio.h>
#include <conio.h>
struct pstruct
{
        int fno;
        int pbit;
}
ptable[10];
int pmsize,lmsize,psize,frame,page,ftable[20],frameno;
void info()
{
        printf("\n\nMEMORY MANAGEMENT USING PAGING\n\n");
        printf("\n\nEnter the Size of Physical memory: ");
        scanf("%d",&pmsize);
        printf("\n\nEnter the size of Logical memory: ");
        scanf("%d",&lmsize);
        printf("\n\nEnter the partition size: ");
        scanf("%d",&psize);
        frame = (int) pmsize/psize;
        page = (int) lmsize/psize;
        printf("\nThe physical memory is divided into %d no.of frames\n",frame);
        printf("\nThe Logical memory is divided into %d no.of pages",page);
}
void assign()
{
        int i;
        for (i=0;i<page;i++)
        {
        ptable[i].fno = -1;
        ptable[i].pbit= -1;
        }
        for(i=0; i<frame;i++)
                ftable[i] = 32555;
        for (i=0;i<page;i++)
        {
        printf("\n\nEnter the Frame number where page %d must be placed: ",i);
                scanf("%d",&frameno);
                ftable[frameno] = i;
                if(ptable[i].pbit == -1)
                {
                        ptable[i].fno = frameno;
```

```
                        ptable[i].pbit = 1;
                }
        }
        getch();
        printf("\n\nPAGE TABLE\n\n");
        printf("PageAddress FrameNo. PresenceBit\n\n");
        for (i=0;i<page;i++)
                printf("%d\t\t%d\t\t%d\n",i,ptable[i].fno,ptable[i].pbit);
        printf("\n\n\n\tFRAME TABLE\n\n");
        printf("FrameAddress PageNo\n\n");
        for(i=0;i<frame;i++)
                printf("%d\t\t%d\n",i,ftable[i]);
}
void cphyaddr()
{
        int laddr,paddr,disp,phyaddr,baddr;
        getch();
        printf("\n\n\n\tProcess to create the Physical Address\n\n");
        printf("\nEnter the Base Address: ");
        scanf("%d",&baddr);
        printf("\nEnter theLogical Address: ");
        scanf("%d",&laddr);

        paddr = laddr / psize;
        disp = laddr % psize;
         if(ptable[paddr].pbit == 1 )
                phyaddr = baddr + (ptable[paddr].fno*psize) + disp;
        printf("\nThe Physical Address where the instruction present: %d",phyaddr);
}
void main()
{
        clrscr();
        info();
        assign();
        cphyaddr();
        }
```

OUTPUT: MEMORY MANAGEMENT USING PAGING

Enter the Size of Physical memory: 16

Enter the size of Logical memory: 8

Enter the partition size: 2

The physical memory is divided into 8 no.of frames

The Logical memory is divided into 4 no.of pages

Enter the Frame number where page 0 must be placed: 5

Enter the Frame number where page 1 must be placed: 6

Enter the Frame number where page 2 must be placed: 7

Enter the Frame number where page 3 must be placed: 2

PAGE TABLE

PageAddress	FrameNo.	PresenceBit
0	5	1
1	6	1
2	7	1
3	2	1

FRAME TABLE

FrameAddress	PageNo
0	32555
1	32555
2	3
3	32555
4	32555
5	0
6	1
7	2

Process to create the Physical Address

Enter the Base Address: 1000

Enter the Logical Address: 3

The Physical Address where the instruction present: 1013

Week 5: Write a program to implement page replacement algorithms (FIFO, Optimal, and LRU).

a) AIM: To implement page replacement algorithms FIFO (First In First Out)

Algorithm

Step 1: Create a queue to hold all pages in memory

Step 2: When the page is required replace the page at the head of the queue

Step 3: Now the new page is inserted at the tail of the queue

/* FIFO Page Replacement Algorithm */

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{
        printf("\n \t\t\t FIFI PAGE REPLACEMENT ALGORITHM");
         printf("\n Enter no.of frames... ");
         scanf("%d",&nof);
         printf("Enter number of reference string..\n");
         scanf("%d",&nor);
         printf("\n Enter the reference string..");
         for(i=0;i<nor;i++)
         scanf("%d",&ref[i]);
         printf("\nThe given reference string:");
         for(i=0;i<nor;i++)
         printf("%4d",ref[i]);
         for(i=1;i<=nof;i++)
         frm[i]=-1;
         printf("\n");
         for(i=0;i<nor;i++)
         {
                 flag=0;
                 printf("\n\t Reference np%d->\t",ref[i]);
                 for(j=0;j<nof;j++)
                 {
                          if(frm[j]==ref[i])
                         {
                                  flag=1;
```

```
                        break;
                    }
                }
                if(flag==0)
                {
                    pf++;
                    victim++;
                    victim=victim%nof;
                    frm[victim]=ref[i];
                    for(j=0;j<nof;j++)
                    printf("%4d",frm[j]);
                }
            }
            printf("\n\n\t\t No.of pages faults...%d",pf);
}
```

OUTPUT:

FIFO PAGE REPLACEMENT ALGORITHM

Enter no.of frames... 4

Enter number of reference string..

6

Enter the reference string..

5 6 4 1 2 3

The given reference string:

..... 5 6 4 1 2 3

Reference np5->	5 -1 -1 -1
Reference np6->	5 6 -1 -1
Reference np4->	5 6 4 -1
Reference np1->	5 6 4 1
Reference np2->	2 6 4 1
Reference np3->	2 3 4 1

No. of pages faults..6

b) AIM: To implement page replacement algorithms

Optimal (The page which is not used for longest time)

Algorithm:

Optimal algorithm

Here we select the page that will not be used for the longest period of time.

OPTIMAL:

Step 1: Create a array

Step 2: When the page fault occurs replace page that will not be used for the longest period of time

/*OPTIMAL(LFU) page replacement algorithm*/

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
          printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
          printf("\n..... ");
          printf("\nEnter the no.of frames");
         scanf("%d",&nof);
         printf("Enter the no.of reference string");
         scanf("%d",&nor);
          printf("Enter the reference string");
         for(i=0;i<nor;i++)
         scanf("%d",&ref[i]);
         clrscr();
         printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
         printf("\n. ..... ");
         printf("\nThe given string");
         printf("\n. .....\n");
         for(i=0;i<nor;i++)
         printf("%4d",ref[i]);
         for(i=0;i<nof;i++)
         {
```

```
                frm[i]=-1;
                optcal[i]=0;
        }
        for(i=0;i<10;i++)
        recent[i]=0;
        printf("\n");
        for(i=0;i<nor;i++)
        {
                flag=0;
                printf("\n\tref no %d ->\t",ref[i]);
                for(j=0;j<nof;j++)
                {
                        if(frm[j]==ref[i])
                        {
                                flag=1;
                                break;
                        }
                }
                if(flag==0)
                {
                        count++;
                        if(count<=nof)
                        victim++;
                        else
                        victim=optvictim(i);
                        pf++;
                        frm[victim]=ref[i];
                        for(j=0;j<nof;j++)
                        printf("%4d",frm[j]);
                }
        }
        printf("\n Number of page faults: %d",pf);
}
int optvictim(int index)
{
        int i,j,temp,notfound;
        for(i=0;i<nof;i++)
        {
                notfound=1;
                for(j=index;j<nor;j++)
                if(frm[i]==ref[j])
                {
                        notfound=0;
```

```
                        optcal[i]=j;
                        break;
                }
                if(notfound==1)
                        return i;
        }
        temp=optcal[0];
        for(i=1;i<nof; i++)
        if(temp<optcal[i])
        temp=optcal[i];
        for(i=0;i<nof;i++)
        if(frm[temp]==frm[i])
        return i;
        return 0;
}
```

OUTPUT:

Enter no.of Frames...3

Enter no.of reference string. 6

Enter reference string..

6 5 4 2 3 1

OPTIMAL PAGE REPLACEMENT ALGORITHM

The given reference string:

……. 6 5 4 2 3 1

Reference NO 6-> 6 -1 -1

Reference NO 5-> 6 5 -1

Reference NO 4-> 6 5 4

Reference NO 2-> 2 5 4

Reference NO 3-> 3 5 4

Reference NO 1-> 1 5 4

No.of page faults...6

C) AIM: To implement page replacement algorithm LRU (Least Recently Used)

Here we select the page that has not been used for the longest period of time.

Algorithm:

Step 1: Create a queue to hold all pages in memory

Step 2: When the page is required replace the page at the head of the queue

Step 3: Now the new page is inserted at the tail of the queue

Step 4: Create a stack

Step 5: When the page fault occurs replace page present at the bottom of the stack

/* LRU Page Replacement Algorithm */

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictim();
void main()
{
        printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
        printf("\n Enter no.of Frames...");
        scanf("%d",&nof);
        printf(" Enter no.of reference string..");
        scanf("%d",&nor);
        printf("\n Enter reference string..");
        for(i=0;i<nor;i++)
        scanf("%d",&ref[i]);
        printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM ");
        printf("\n\t The given reference string:");
        printf("\n….....");
        for(i=0;i<nor;i++)
        printf("%4d",ref[i]);
        for(i=1;i<=nof;i++)
        {
                frm[i]=-1;
                 lrucal[i]=0;
        }
        for(i=0;i<10;i++)
        recent[i]=0;
        printf("\n");
```

```
        for(i=0;i<nor;i++)
        {
                flag=0;
                printf("\n\t Reference NO %d->\t",ref[i]);
                for(j=0;j<nof;j++)
                {
                 if(frm[j]==ref[i])
                {
                        flag=1;
                         break;
                }
                }
                if(flag==0)
                {
                        count++;
                        if(count<=nof)
                        victim++;
                         else
                        victim=lruvictim();
                        pf++;
                        frm[victim]=ref[i];
                        for(j=0;j<nof;j++)
                        printf("%4d",frm[j]);
                }
                recent[ref[i]]=i;
          }
          printf("\n\n\t No.of page faults...%d",pf);
          getch();
}
int lruvictim()
{
        int i,j,temp1,temp2;
        for(i=0;i<nof;i++)
        {
                temp1=frm[i];
                lrucal[i]=recent[temp1];
          }
           temp2=lrucal[0];
          for(j=1;j<nof;j++)
         {
                 if(temp2>lrucal[j])
                 temp2=lrucal[j];
          }
```

```
        for(i=0;i<nof;i++)
        if(ref[temp2]==frm[i])
        return i;
        return 0;
}
```

OUTPUT:

LRU PAGE REPLACEMENT ALGORITHM

Enter no.of Frames…3

Enter no.of reference string. 6

Enter reference string..

6 5 4 2 3 1

LRU PAGE REPLACEMENT ALGORITHM

The given reference string:

……. 6 5 4 2 3 1

Reference NO 6-> 6 -1 -1

Reference NO 5-> 6 5 -1

Reference NO 4-> 6 5 4

Reference NO 2-> 2 5 4

Reference NO 3-> 2 3 4

Reference NO 1-> 2 3 1

No.of page faults...6

Week 6: Write a C program to simulate the following file allocation strategies.

a) Sequential b) Indexed c) Linked

a) AIM: **Write a C Program to implement Sequential File Allocation method.**

Algorithm:

Step 1: Start the program.
Step 2: Get the number of memory partition and their sizes.
Step 3: Get the number of processes and values of block size for each process.
Step 4: First fit algorithm searches all the entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.
Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates if.
Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.
Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.
Step 8: Stop the program.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,i,j,b[20],sb[20],t[20],x,c[20][20];
 clrscr();
 printf("Enter no.of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
        printf("Enter no. of blocks occupied by file%d",i+1);
        scanf("%d",&b[i]);
        printf("Enter the starting block of file%d",i+1);
        scanf("%d",&sb[i]);
        t[i]=sb[i];
        for(j=0;j<b[i];j++)
                c[i][j]=sb[i]++;
        }
 printf("Filename\tStart block\tlength\n");
 for(i=0;i<n;i++)
```

```
        printf("%d\t %d \t%d\n",i+1,t[i],b[i]);
 printf("Enter file name:");
 scanf("%d",&x);
 printf("File name is:%d",x);
 printf("length is:%d",b[x-1]);
 printf("blocks occupied:");
 for(i=0;i<b[x-1];i++)
        printf("%4d",c[x-1][i]);
 getch();
}
```

OUTPUT:

Enter no.of files: 2

Enter no. of blocks occupied by file1 4

Enter the starting block of file1 2

Enter no. of blocks occupied by file2 10

Enter the starting block of file2 5

Filename	Start block	length
1	2	4
2	5	10

Enter file name: 1

File name is:1 length is:4 blocks occupied 2 3 4 5

b) AIM: **Write a C Program to implement Indexed File Allocation method.**

Algorithm:

Step 1: Start.
Step 2: Let n be the size of the buffer
Step 3: check if there are any producer
Step 4: if yes check whether the buffer is full
Step 5: If no the producer item is stored in the buffer
Step 6: If the buffer is full the producer has to wait
Step 7: Check there is any cosumer.If yes check whether the buffer is empty
Step 8: If no the consumer consumes them from the buffer
Step 9: If the buffer is empty, the consumer has to wait.
Step 10: Repeat checking for the producer and consumer till required
Step 11: Terminate the process.

PROGRAM

```
#include<stdio.h>
main()
{
 int n,m[20],i,j,sb[20],s[20],b[20][20],x;
printf("Enter no. of files:");
scanf("%d",&n);
 for(i=0;i<n;i++)
 {      printf("Enter starting block and size of file%d:",i+1);
        scanf("%d%d",&sb[i],&s[i]);
        printf("Enter blocks occupied by file%d:",i+1);
        scanf("%d",&m[i]);
        printf("enter blocks of file%d:",i+1);
        for(j=0;j<m[i];j++)
                scanf("%d",&b[i][j]);
 } printf("\nFile\t index\tlength\n");
 for(i=0;i<n;i++) {
        printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
 }
}
```

OUTPUT:

```
Enter no. of files:3
Enter starting block and size of file1:
2
6
Enter blocks occupied by file1:5
enter blocks of file1:2 3 4 7 9
Enter starting block and size of file2:7
2
Enter blocks occupied by file2:1
enter blocks of file2:3
Enter starting block and size of file3:6
3
Enter blocks occupied by file3:2
enter blocks of file3:8
9
File     index  length
1       2       5
2       7       1
3       6       2
```

C) AIM: **Write a C Program to implement Linked File Allocation method.**

Algorithm

Step 1: Create a queue to hold all pages in memory
Step 2: When the page is required replace the page at the head of the queue
Step 3: Now the new page is inserted at the tail of the queue
Step 4: Create a stack
Step 5: When the page fault occurs replace page present at the bottom of the stack
Step 6: Stop the allocation.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct file
{
 char fname[10];
 int start,size,block[10];
}f[10];
main()
{
        int i,j,n;
       printf("Enter no. of files:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter file name:");
                scanf("%s",&f[i].fname);
                printf("Enter starting block:");
                scanf("%d",&f[i].start);
                f[i].block[0]=f[i].start;
                printf("Enter no.of blocks:");
                scanf("%d",&f[i].size);
                printf("Enter block numbers:");
                for(j=1;j<=f[i].size;j++)
                {
                        scanf("%d",&f[i].block[j]);
                }
        }
        printf("File\tstart\tsize\tblock\n");
        for(i=0;i<n;i++)
        {
```

```
                printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
                for(j=1;j<=f[i].size-1;j++)
                printf("%d--->",f[i].block[j]);
                printf("%d",f[i].block[j]);
                printf("\n");
        }
}
```

OUTPUT

```
Enter no. of files:2
Enter file name:a
Enter starting block:3
Enter no.of blocks:3
Enter block numbers:3
4
5
Enter file name:b
Enter starting block:6
Enter no.of blocks:4
Enter block numbers:6
7
8
9
File    start   size    block
a       3       3       3--->4--->5
b       6       4       6--->7--->8--->9
```

Week 7: Write a program to implement Banker's algorithm for deadlock avoidance

AIM: To implement deadlock avoidance & Prevention by using Banker's Algorithm.

Banker's Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures

- n-Number of process, m-number of resource types.
- Available: Available[j]=k, k – instance of resource type Rj is available.
- Max: If max[i, j]=k, Pi may request at most k instances resource Rj.
- Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj
- Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj,
  Need[I, j]=Max[I, j]-Allocation[I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
   - Finish[i] =False
   - Need<=Work

   If no such I exists go to step 4.

3. work=work+Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;
   Available=Available-Request I;

Allocation I =Allocation+Request I;

Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

Algorithm:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.

/* BANKER'S ALGORITHM */

```
#include<stdio.h>
#include<conio.h>
struct da
{
        int max[10],a1[10],need[10],before[10],after[10];
 }p[10];
 void main()
 {
        int i,j,k,l,r,n,tot[10],av[10],cn=0,cz=0,temp=0,c=0;
        clrscr();
        printf("\n ENTER THE NO. OF PROCESSES:");
        scanf("%d",&n);
        printf("\n ENTER THE NO. OF RESOURCES:");
        scanf("%d",&r);
        for(i=0;i<n;i++)
        {
                printf("PROCESS %d \n",i+1);
                for(j=0;j<r;j++)
                {
                        printf("MAXIMUM VALUE FOR RESOURCE %d:",j+1);
                        scanf("%d",&p[i].max[j]);
                }
                for(j=0;j<r;j++)
                {
                        printf("ALLOCATED FROM RESOURCE %d:",j+1);
                        scanf("%d",&p[i].a1[j]);
                        p[i].need[j]=p[i].max[j]-p[i].a1[j];
                }
         }
        for(i=0;i<r;i++)
        {
                printf("ENTER TOTAL VALUE OF RESOURCE %d:",i+1);
                scanf("%d",&tot[i]);
        }
        for(i=0;i<r;i++)
        {
                for(j=0;j<n;j++)
                 temp=temp+p[j].a1[i];
                 av[i]=tot[i]-temp;
                 temp=0;
        }
```

```
 printf("\n\t RESOURCES ALLOCATED NEEDED TOTAL  AVAIL");
for(i=0;i<n;i++)
{
          printf("\n P%d \t",i+1);
         for(j=0;j<r;j++)
         printf("%d",p[i].max[j]);
         printf("\t");
         for(j=0;j<r;j++)
         printf("%d",p[i].a1[j]);
         printf("\t");
         for(j=0;j<r;j++)
         printf("%d",p[i].need[j]);
         printf("\t");
         for(j=0;j<r;j++)
         {
                  if(i==0)
                  printf("%d",tot[j]);
         }
          printf("    ");
          for(j=0;j<r;j++)
         {
                  if(i==0)
                  printf("%d",av[j]);
         }
          }
         printf("\n\n\t AVAIL BEFORE\T AVAIL AFTER ");
          for(l=0;l<n;l++)
         {
                 for(i=0;i<n;i++)
                  {
                  for(j=0;j<r;j++)
                  {
                           if(p[i].need[j] >av[j])
                           cn++;
                           if(p[i].max[j]==0)
                           cz++;
                  }
                  if(cn==0 && cz!=r)
                 {
                          for(j=0;j<r;j++)
                          {
                                   p[i].before[j]=av[j]-p[i].need[j];
                                   p[i].after[j]=p[i].before[j]+p[i].max[j];
```

```
                        av[j]=p[i].after[j];
                        p[i].max[j]=0;
                }
                 printf("\n P %d \t",i+1);
                for(j=0;j<r;j++)
                printf("%d",p[i].before[j]);
                printf("\t");
                for(j=0;j<r;j++)
                printf("%d",p[i].after[j]);
                cn=0;
                cz=0;
                c++;
                 break;
        }
        else
        {
                 cn=0;cz=0;
         }
        }
        }
        if(c==n)
        printf("\n THE ABOVE SEQUENCE IS A SAFE SEQUENCE");
        else
         printf("\n DEADLOCK OCCURED");
}
```

OUTPUT:

```
//TEST CASE 1:
 ENTER THE NO. OF PROCESSES:4
ENTER THE NO. OF RESOURCES:3
PROCESS 1
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:1
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:0
PROCESS 2
MAXIMUM VALUE FOR RESOURCE 1:6
MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:3
ALLOCATED FROM RESOURCE 1:5
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1
PROCESS 3
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:4
ALLOCATED FROM RESOURCE 1:2
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1
PROCESS 4
MAXIMUM VALUE FOR RESOURCE 1:4
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:0
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:2
ENTER TOTAL VALUE OF RESOURCE 1:9
ENTER TOTAL VALUE OF RESOURCE 2:3
ENTER TOTAL VALUE OF RESOURCE 3:6
```

RESOURCES	ALLOCATED	NEEDED	TOTAL	AVAIL
P1	322	100	222	936 112
P2	613	511	102	
P3	314	211	103	
P4	422	002	420	

```
 AVAIL BEFORE AVAIL AFTER
 P 2          010    623
 P 1          401    723
 P 3          620    934
 P 4          514    936
THE ABOVE SEQUENCE IS A SAFE SEQUENCE
//TEST CASE:2
 ENTER THE NO. OF PROCESSES:4
ENTER THE NO. OF RESOURCES:3
PROCESS 1
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:1
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:1
PROCESS 2
MAXIMUM VALUE FOR RESOURCE 1:6
MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:3
ALLOCATED FROM RESOURCE 1:5
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1
PROCESS 3
MAXIMUM VALUE FOR RESOURCE 1:3
MAXIMUM VALUE FOR RESOURCE 2:1
MAXIMUM VALUE FOR RESOURCE 3:4
ALLOCATED FROM RESOURCE 1:2
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:2
PROCESS 4
MAXIMUM VALUE FOR RESOURCE 1:4
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:0
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:2
ENTER TOTAL VALUE OF RESOURCE 1:9
ENTER TOTAL VALUE OF RESOURCE 2:3
ENTER TOTAL VALUE OF RESOURCE 3:6
    RESOURCES ALLOCATED NEEDED TOTAL  AVAIL
 P1    322     101       221    936   110
 P2    613     511       102
```

P3 314 212 102
P4 422 002 420
AVAIL BEFORE AVAIL AFTER
DEADLOCK OCCURED

Week : 8

Design and Implement the data link layer framing methods such as character stuffing and bit stuffing

(a) AIM: **To write a program to implement character stuffing.**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<process.h>
void main()
  {
inti=0,j=0,n,pos;
char a[20],b[50],u[50],ch;
clrscr();
printf("enter string\n");
scanf("%s",&a);
n=strlen(a);
b[0]='x';
b[1]='x';
j=j+2;
while(a[i]!='\0')
        {
            if(a[i]=='x' && a[i+1]=='x')
                {
                    b[j]='x';
                    b[j+1]='x';
                    j=j+2;
                }
            b[j]=a[i];
            i++;
            j++;
          }
        b[j]='x';
        b[j+1]='x';
        b[j+2]='\0';
        printf("\nframe after stuffing:\n");
        printf("%s",b);
          //unstuffing
        i=0;j=0;
        while(b[i]!='\0')
          {
```

```
if(b[i]=='x' && b[i+1]=='x')
    {
u[j]=b[i+1];
i=i+2;
    }
u[j]=b[i];
i++;
j++;

   }
u[j]='\0';
printf("\n unstuffed data is %s",u);
    }
```

OUTPUT:

Enter the input character string:

acdxxbfh

the stuffed character string is:

xxacdxxxxbfhxx

the unstuffed character string is :

acdxxbfh

(b) AIM: **To write a program to implement Bit-Stuffing**

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 100
void main( )
{
char in[MAXSIZE];
char stuff[MAXSIZE];
char unstuff[MAXSIZE];
int count=0,j=0,i=0;
printf("enter the input character string (0ës & 1ës only):\n");
scanf("%s",in);
while(in[i]!='\0')
  {
if(in[i]=='0')
    {
stuff[j]=in[i];
i++;
j++;
    }
else
    {
while(in[i]=='1' && count!=5)
     {
          count++;
          stuff[j]=in[i];
          i++;
          j++;
     }
if(count==5)
     {
stuff[j]='0';
          j++;
     }
count=0;
    }
  }
stuff[j]='\0';
printf("\nthe stuffed character string is");
printf("\n%s",stuff);
i=0;
j=0;
```

```
while(stuff[i]!='\0')
  {
if(stuff[i]=='0')
   {
unstuff[j]=stuff[i];

i++;
j++;
   }
else
   {
while(stuff[i]=='1' && count!=5)
     {
          count++;
          unstuff[j]=stuff[i];
          i++;
          j++;
     }
if(count==5)
     {
          i++;
     }
count=0;
   }
  }
unstuff[j]='\0';
printf("\nthe unstuffed character string is");
printf("\n%s\n",unstuff);
}
```

INPUT/OUTPUT:

enter the input character string:
10111111011
The stuffed character string is:
101111101011
The unstuffed character string is:
10111111011

Week: 9

Implementation of Hamming code algorithm

Program:

```
#include<stdio.h>
#include<string.h>
char XOR(char a,char b,char c,char d);
void main()
{
        char data[10],r[3],syn[10],code[10];
        printf("this works for message of 4bits in size \n");
        gets(data);
        printf("\nredundant bits are:");
        r[2]=data[1]^data[2]^data[3];
        r[1]=data[0]^data[1]^data[2];
        r[0]=data[0]^data[2]^data[3];
        r[3]='\0';
        puts(r);
        strcat(data,r);
        printf("\n%s",data);
        printf("\nenter receiver's side code word\n");
        gets(code);
        printf("syndrome bits are\n");
        fflush(stdin);
        syn[2]=XOR(code[1],code[2],code[3],code[6]);
        syn[1]=XOR(code[0],code[1],code[2],code[5]);
        syn[0]=XOR(code[0],code[2],code[3],code[4]);
        syn[3]='\0';
        printf("%s",syn);
        if(syn[0]=='0'&&syn[1]=='0'&&syn[2]=='0')
                printf("No Error");
        else if(syn[0]=='0'&&syn[1]=='0'&&syn[2]=='1')
        printf("error is in code[6]or Q0");
        else if(syn[0]=='0'&&syn[1]=='1'&&syn[2]=='0')
                printf("error is in code[5] orQ1");
        else if(syn[0]=='0'&&syn[1]=='1'&&syn[2]=='1')
                printf("error is in code[1] orB2");
        else if(syn[0]=='1'&&syn[1]=='0'&&syn[2]=='0')
                printf("error is in code[4] orQ2");
        else if(syn[0]=='1'&&syn[1]=='0'&&syn[2]=='1')
                printf("error is in code[3] orB0");
```

```
        else if(syn[0]=='1'&&syn[1]=='1'&&syn[2]=='0')
                printf("error is in code[0] orB3");
        else
             printf("error is in code[2] orB1");
             }
 char XOR(char a,char b,char c,char d)
    {
                char x;
                x= a^b^c;
                if(x==d)
                        return '0';
                else
                        return '1';
    }
```

Output:

```
This works for message of 4 bits size
1001
Redundant bits are:011
1001011
Enter the receivers side code word
1101011
Syndrome bits are
011
Error is in code[1] or B2
```

Week : 10

Implement on a data set of characters the three CRC polynomials

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[20],c[20],d[20],aux[20],r,i,j,l,in,flag,choice,key;
clrscr();
printf("ENTER THE SIZE OF DATA WORD :\n ");
scanf("%d",&l);
printf("ENTER THE DATA WORD : \n");
for(i=0; i<l; i++)
        scanf("%d",&a[i]);
printf("ENTER THE NUMBER OF REDUNDANT BITS : \n");
scanf("%d",&r);
printf("ENTER THE DIVISOR :\n ");
for(i=0; i<(r+1); i++)
        scanf("%d",&d[i]);
for(i=0; i<(l+r); i++)
    {
        if(i<l) c[i]=a[i];
        else c[i]=0;
    }
printf(" ... GENERATOR MODULE ... \n");
printf("THE INTERMEDIATE CODE WORD IS :\n ");
for(i=0; i<(l+r); i++)
        printf("%d ",c[i]);
printf("\n");
for(i=0; i<(l+r); i++)
        aux[i]=c[i];
for(i=0; i<l; i++)
    {
        in=1;
        if(c[i]==1)
          {
                for(j=i+1; j<(i+l); j++)
                {
                c[j]=c[j]^d[in];
                in++;
                }
          }
```

```
        else

          {
                for(j=i+1; j<(i+l); j++)
                c[j]=c[j]^0;
          }
    }
printf("THE REMAINDER AFTER DIVISION IS :\n ");
for(i=l; i<(l+r); i++)
        printf("%d ",c[i]);
printf("\nSENDER CODE WORD :\n");
for(i=0; i<(l+r); i++)
    {
        if(i<l) c[i]=aux[i];
        printf("%d ",c[i]);
    }
printf("\nPRESS 1 IF YOU WANT TO CHANGE A BIT OR 0 TO CONTINUE :\n ");
scanf("%d",&choice);
if(choice==1)
    {
        printf("ENTER THE BIT YOU WANT TO CHANGE :\n");
        scanf("%d",&key);
        for(i=0; i<(l+r); i++)
          {
                if(i==(key-1))
                {
                if(c[key-1]==0) c[key-1]=1;
                else c[key-1]=0;
                }
          }
        printf("CODE WORD AFTER ERROR :\n ");
        for(i=0; i<(l+r); i++)
                printf("%d ",c[i]);
    }
elseprintf("\nNO ERROR INSERTED IN CODE WORD...\n");
printf(" ... CHECKER MODULE ...\n ");
for(i=0; i<l; i++)
    {
        in=1;
        if(c[i]==1)
         {
                for(j=i+1; j<(i+l); j++)
                {
```

```
                c[j]=c[j]^d[in];
                in++;
                }
          }
        else
          {
                for(j=i+1; j<(i+l); j++)
                c[j]=c[j]^0;
          }
    }
printf("\nTHE SYNDROME ARRAY IS :\n ");
for(i=l; i<(l+r); i++)
        printf("%d ",c[i]);
flag=0;
for(i=l; i<(l+r); i++)
    {
        if(c[i]!=0)
          {
                flag=1;
                break;
          }
    }
if(flag==1) printf("ERROR DETECTED !!!\n");
elseprintf("NO ERROR FOUND\n");
}
```

Output:

```
ENTER THE SIZE OF DATA WORD :
 4
ENTER THE DATA WORD :
1
0
0
1

ENTER THE NUMBER OF REDUNDANT BITS :
3
ENTER THE DIVISOR :
 1
0
1
1
 ... GENERATOR MODULE ...
THE INTERMEDIATE CODE WORD IS :
 1 0 0 1 0 0 0
THE REMAINDER AFTER DIVISION IS :
 1 1 0
SENDER CODE WORD :
1 0 0 1 1 1 0
PRESS 1 IF YOU WANT TO CHANGE A BIT OR 0 TO CONTINUE :0
NO ERROR INSERTED IN CODE WORD...
 ... CHECKER MODULE ...

THE SYNDROME ARRAY IS :
 0 0 0 NO ERROR FOUND
```

Week 11:

Implement Dijkstra's algorithm to compute the Shortest path through a graph.

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
 void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{
   int G[MAX][MAX],i,j,n,u;
   printf("Enter no. of vertices:");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix:\n");
      for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         scanf("%d",&G[i][j]);
      printf("\nEnter the starting node:");
   scanf("%d",&u);
   dijkstra(G,n,u);
      return 0;
}
 Void dijkstra(int G[MAX][MAX],int n,int startnode)
{
   int cost[MAX][MAX],distance[MAX],pred[MAX];
   int visited[MAX],count,mindistance,nextnode,i,j;

   //pred[] stores the predecessor of each node
   //count gives the number of nodes seen so far
   //create the cost matrix
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         if(G[i][j]==0)
            cost[i][j]=INFINITY;
         else
            cost[i][j]=G[i][j];
      //initialize pred[],distance[] and visited[]
   for(i=0;i<n;i++)
   {
      distance[i]=cost[startnode][i];
      pred[i]=startnode;
      visited[i]=0;
```

```
    }
      distance[startnode]=0;
    visited[startnode]=1;
    count=1;
      while(count<n-1)
    {
       mindistance=INFINITY;
        //nextnode gives the node at minimum distance
       for(i=0;i<n;i++)
          if(distance[i]<mindistance&&!visited[i])
          {
             mindistance=distance[i];
             nextnode=i;
          }
           //check if a better path exists through nextnode
          visited[nextnode]=1;
          for(i=0;i<n;i++)
             if(!visited[i])
                if(mindistance+cost[nextnode][i]<distance[i])
                {
                   distance[i]=mindistance+cost[nextnode][i];
                   pred[i]=nextnode;
                }
       count++;
    }
    //print the path and distance of each node
    for(i=0;i<n;i++)
       if(i!=startnode)
       {
          printf("\nDistance of node%d=%d",i,distance[i]);
          printf("\nPath=%d",i);
           j=i;
          do
          {
             j=pred[j];
             printf("<-%d",j);
          }while(j!=startnode);
    }
}
```

Output:

```
Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
Path=4<-2<-3<-0
```

Week 12:

Take an example subnet graph with weights indicating delay between nodes. Construct Routing table art each node using Distance Vector Routing Algorithm.

```
#include<stdio.h>
#include<stdlib.h>
int d[10][10], via[10][10];
int main()
{
inti,j, k, n, g[10][10];
printf("\nEnter the no of nodes : ");
scanf("%d", &n);
for(i = 0; i<n; i++)
 {
printf("Enter the record for route %c \n", i+97);
for(j = 0; j<n;j++)
  {
printf("(%c : %c) :: ", i+97, j+97);
scanf("%d", &g[i][j]);
if(g[i][j] != 999)
d[i][j] = 1;
  }
 }
for(i = 0; i<n; i++)
for(j = 0; j<n;j++)
via[i][j] = i;
for(i = 0; i<n; i++)
 {
for(j = 0; j<n;j++)
if(d[i][j] == 1)
for(k = 0; k<n; k++)
if(g[i][j] + g[j][k] < g[i][k])
          {
g[i][k] = g[i][j] + g[j][k];
via[i][k] = j;
          }
 }
for(i = 0; i<n; i++)
 {
printf("cost of route %c :\n",i+97);
for(j = 0; j<n;j++)
printf("%c : %d via %c \n", j+97, g[i][j], via[i][j]+ 97);
 }
return 0; }
```

Output:

```
Enter the no of nodes : 4
Enter the record for route a
(a : a) :: 0
(a : b) :: 2
(a : c) :: 3
(a : d) :: 999
Enter the record for route b
(b : a) :: 2
(b : b) :: 0
(b : c) :: 1
(b : d) :: 999

Enter the record for route c
(c : a) :: 3
(c : b) :: 1
(c : c) :: 0
(c : d) :: 2
Enter the record for route d
(d : a) :: 999
(d : b) :: 999
(d : c) :: 2
(d : d) :: 0
cost of route a :
a : 0 via a
b : 2 via a
c : 3 via a
d : 5 via c
cost of route b :
a : 2 via b
b : 0 via b
c : 1 via b
d : 3 via c
cost of route c :
a : 3 via c
b : 1 via c
c : 0 via c
d : 2 via c
cost of route d :
a : 5 via c
b : 3 via c
c : 2 via d
d : 0 via d
```

Week 13: Write a C-Program to execute Broadcast tree

```
#include<stdio.h>
int a[10][10],n;
void adj(int);
void main()
{
Int i,j,root;
printf("enter no of nodes:");
scanf("%d",&n);
printf("\nenter adjacent matrix\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
 {
printf("enter connecting of %d->%d::",i,j);
scanf("%d",&a[i][j]);
}
printf("\nenter root node:");
scanf("%d",&root);
adj(root);
}
voidadj(int k)
{
inti,j;
printf("adjacent node of root node::\n");
printf("%d->",k);
for(j=1;j<=n;j++)
{
if(a[k][j]==1||a[j][k]==1)
printf("%d->",j);
}
//printf("\n");
for(i=1;i<=n;i++)
{
if((a[k][j]==0)&&(a[i][k]==0)&&(i!=k))
printf("%d->",i);
}
}
```

Output:

```
enter no of nodes:5
enter adjacent matrix
enter connecting of 1->1::0
enter connecting of 1->2::1
enter connecting of 1->3::1
enter connecting of 1->4::1
enter connecting of 1->5::0
enter connecting of 2->1::1
enter connecting of 2->2::0
enter connecting of 2->3::1
enter connecting of 2->4::0
enter connecting of 2->5::1
enter connecting of 3->1::1
enter connecting of 3->2::1
enter connecting of 3->3::0
enter connecting of 3->4::0
enter connecting of 3->5::1
enter connecting of 4->1::1
enter connecting of 4->2::0
enter connecting of 4->3::0
enter connecting of 4->4::0
enter connecting of 4->5::0
enter connecting of 5->1::0
enter connecting of 5->2::1
enter connecting of 5->3::1
enter connecting of 5->4::0
enter connecting of 5->5::0
enter root node:4
adjacent node of root node::
4->1->2->3->5->
```

Vidya Jyothi Institute of Technology

(Approved by AICTE, New Delhi, Accredited by NAAC, Permanently Affiliated to JNTUH, Hyderabad)

An Autonomous Institution

Aziznagar Gate, Chilkur Balaji Road,

Hyderabad – 500075, Telangana, India

Department of Information Technology

Rubrics for Continuous Assessment

Course: Computer networks & operating Systems **Evaluator:** D. Sravanthi

Date: **Overall Score: 15M**

Objects	5 Out Standing	4 Good	3 Moderate	2 Normal	1 Satisfied	0 Not Satisfied
Observation-5M						
Record-5M						
Execution -5M						

HOD-IT

Head of the Department
Dept. of Information Technology
Vidya Jyothi Institute of Technology
Aziz Nagar Gate, C.B. Post
Hyderabad - 500 075, Telangana

Vidya Jyothi Institute of Technology

(Approved by AICTE, New Delhi, Accredited by NAAC, Permanently Affiliated to JNTUH, Hyderabad)

An Autonomous Institution

Aziznagar Gate, ChilkurBalaji Road,

Hyderabad – 500075, Telangana, India

Department of Information Technology

Lab Assessment:

Continuous Assessment Week Wise	**15M**
Lab Internal Examination	**10M**
Total	**25M**

HOD-IT

Head of the Department
Dept. of Information Technology
Vidya Jyothi Institute of Technology
Aziz Nagar Gate, C.B. Post
Hyderabad - 500 075, Telangana

IMPLEMENTATION

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

Aziz Nagar Gate, C.B Post, Hyderabad – 500075

DEPARTMENT OF INFORMATION TECHNOLOGY

Continuous Lab Assessment Sheet

Academic Year: 2021-22
Course Name: CN & OS Lab

Class: III - I

Regulation: R19
Course Code: _____

S.No.	Roll No.	Date : 7/10				Date : 21/10				Date : 28/10				Date : 11/11				Date : 18/11				Date : 9/12				Date : 16/12				Date : 23/12			
		O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T
1	19911A1201	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
2	19911A1202	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
3	19911A1203	←			→	←			→	5	5	5	15	5	5	5	15	5	5	5	15	→				5	3	5	13	5	4	5	14
4	19911A1204	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	4	5	14	←			→	5	5	5	15	5	5	5	15
5	19911A1205	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	←			→	→				5	5	5	15	5	5	5	15
6	19911A1206	~~5~~	~~5~~	~~5~~	~~15~~	←			→	5	5	5	15	5	5	5	15	5	4	4	13	5	5	3	13	5	5	5	15	5	5	5	15
7	19911A1207	15	5	5	15	5	5	5	15	5	5	~~4~~	~~14~~	5	5	5	15	5	5	4	14	5	5	4	14	5	5	5	15	5	5	5	15
8	19911A1208	←			→	←			→	5	5	5	15	←			→	←			→	5	4	14		5	5	5	15	5	5	5	15
9	19911A1209	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	4	14	5	5	5	15	5	5	5	15	5	5	5	15
10	19911A1210	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
11	19911A1211	←			→	5	5	5	15	5	5	~~4~~	~~14~~	5	5	5	15	5	5	4	14	5	5	5	15	5	5	5	15	5	4	4	13
12	19911A1212	←			→	←			→	5	5	5	15	←			→	←			→	5	5	15		5	5	4	14	5	4	5	14
13	19911A1213	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	4	14	5	5	5	15
14	19911A1214	~~5~~	~~5~~	~~5~~	~~15~~	←			→	5	5	5	15	5	5	5	15	5	4	5	14	5	5	5	15	5	5	5	15	5	4	5	14
15	19911A1215	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	←			→	5	5	5	15	5	4	5	14	5	5	5	15
16	19911A1216	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
17	19911A1217	←			→	5	4	5	14	5	5	5	15	←			→	5	5	5	15	5	4	5	14	5	5	5	15	5	5	5	15
18	19911A1218	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	4	14	5	4	5	14	5	5	5	15	5	5	5	15
19	19911A1219	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	→				5	5	5	15	5	4	5	14
20	19911A1220	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
21	19911A1221	←			→	5	5	5	15	5	5	5	15	5	5	5	15	←			→	5	5	5	15	5	4	4	13	5	5	5	15
22	19911A1222	5	4	5	14	5	5	5	15	5	5	5	15	←			→	5	4	4	13	5	5	5	15	5	4	5	14	5	5	5	15
23	19911A1223	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	4	14	5	5	5	15
24	19911A1224	←			→	←			→	5	5	5	15	5	5	5	15	5	4	5	15	5	5	5	15	5	5	5	15	5	4	5	14
25	19911A1225	←			→	5	5	05	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	4	14	5	5	3	13	5	4	5	14

O - Observation(5M), R - Record(5M), E - Execution(5M), T - Total Marks (15M)

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY
Aziz Nagar Gate, C.B Post, Hyderabad – 500075
DEPARTMENT OF INFORMATION TECHNOLOGY

Continuous Lab Assessment Sheet

S.No.	Roll No.	Date : 7/10/21				Date : 21/10/21				Date : 28/10/21				Date : 11/11/21				Date : 18/11/21				Date : 2/12/21				Date : 16/12/21				Date : 23/12			
		O	R	E	T	O	R	O	R	E	T	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T
26	19911A1227	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	4	14	5	5	4	14
27	19911A1228	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	4	4	13
28	19911A1229	←			→	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	4	5	14	5	5	5	15	5	5	5	15
29	19911A1230	5	5	5	15	5	5	5	15	5	4	5	14	5	5	5	15	←			→	5	4	5	14	5	4	4	13	5	4	4	13
30	19911A1231	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	4	5	14	5	4	5	14
31	19911A1232	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	4	5	14	5	4	5	14
32	19911A1233	←			→	←			→	5	3	5	13	←			→	5	5	5	15	←			→	5	5	3	13	5	3	2	10
33	19911A1234	5	5	5	15	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
34	19911A1235	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
35	19911A1236	5	5	5	15	←			→	←			→	←			→	←			→	5	5	5	15	5	2	2	9	5	3	2	10
36	19911A1237	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5			→	5	5	5	15	5	5	4	14	5	5	3	13
37	19911A1238	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	←			→	5	5	5	15
38	19911A1239	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	4	5	14	5	5	5	15	5	5	5	15
39	19911A1240	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
40	19911A1241	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
41	19911A1242	5	5	5	15	←			→	5	5	5	15	5	5	5	15	←			→	4	5	5	14	5	4	5	14	5	4	5	14
42	19911A1243	5	5	5	15	←			→	5	5	5	15	5	5	5	15	←			→	5	5	5	15	5	4	5	14	5	5	5	15
43	19911A1244	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
44	19911A1245	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	5	5	15
45	19911A1246	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	5	5	15
46	19911A1247	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
47	19911A1248	5	5	5	15	←			→	5	4	5	14	5	4	5	14	←			→	5	5	5	15	5	5	4	14	5	5	5	15
48	19911A1249	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	4	14	5	4	3	12
49	19911A1250	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
50	19911A1251	5	5	5	15	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15
51	19911A1252	←			→	←			→	5	5	5	15	5	5	5	15	5	5	5	15	5	5	2	12	5	3	2	10	5	3	3	11

O - Observation(5M), R - Record(5M), E - Execution(5M), T - Total Marks (15M)

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

Aziz Nagar Gate, C.B Post, Hyderabad – 500075

DEPARTMENT OF INFORMATION TECHNOLOGY

Continuous Lab Assessment Sheet

S.No	Roll No.	Date : 7/10				Date : 21/10				Date : 28/10				Date : 11/11				Date : 18/11				Date : 2/12				Date : 16/12				Date : 23/12			
		O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T	O	R	E	T
52	19911A1253									5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15	5	4	4	13	5	5	4	14
53	19911A1254	5	4	5	14	5	5	5	15	5	5	5	15	5	4	5	14					5	5	4	14	5	4	5	14	5	4	5	14
54	19911A1255	5	5	5	15	5	5	5	15	5	5	5	15	5	4	5	14	5	4	5	14	5	5	5	15	5	5	5	15	5	5	5	15
55	19911A1256	5	5	5	15	5	5	5	15	5	5	5	15	5	5	5	15					5	5	5	15	5	5	5	15	5	5	5	15
56	19911A1257					5	5	5	15					5	4	5	14	5	5	5	15	5	5	5	15	5	4	4	13	5	4	5	14
57	19911A1258									5	4	4	13	5	5	5	15	5	5	5	15	5	5	5	15	5	4	2	11	5	4	3	12
58	19911A1259	5	5	5	15					5	4	4	13	5	4	5	14	5	5	5	15	5	4	5	14					5	4	4	13
59	19911A1260	5	5	5	15					5	4	4	13	5	5	5	15	5	5	5	15	5	5	4	14	5	5	5	15	5	4	4	13
60	20915A1201					5	5	5	15	5	5	5	15	5	5	4	14	5	5	5	15	5	5	5	15	15	5	4	14	5	5	5	15
61	20915A1202					5	5	5	15	5	4	5	14	5	5	5	15	5	4	5	14	5	5	5	15	5	4	5	14	5	4	4	13
62	20915A1203									5	5	5	15	5	4	5	14	5	4	5	14	5	4	5	14	5	5	5	15	5	4	4	13
63	20915A1204									5	5	5	15	5	5	5	15	5	4	3	12	5	5	5	15					5	4	3	12

O – Observation (5M), R - Record(5M), E - Execution(5M), T - Total Marks (15M)

(Course Instructor)