OPERATING SYSTEMS & COMPILER DESIGN

LAB MANUAL

INDEX

Ex. No:1(a) ROUND ROBIN SCHEDULING

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

(a) Average waiting time = Total waiting Time / Number of process

(b) 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 8

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

Ex. No: 1(b) SJF SCHEDULING

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

(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

(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
```

Ex. No: 1(c) FCFS SCHEDULING

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

(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

(e) Average waiting time = Total waiting Time / Number of process

(f) 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

Ex. No: 1(d) PRIORITY SCHEDULING

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

(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

/* 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;
        clrscr();
        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);
        getch();
}
```

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
```

Exp no:2(a) SEQUENTIAL FILE ALLOCATION

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: rajesh
 File name is:12803 length is:0blocks occupied
```

Exp no:2(b) INDEXED FILE ALLOCATION

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>
#include<conio.h>
main()
{
 int n,m[20],i,j,sb[20],s[20],b[20][20],x;
 clrscr();
 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]);
 }printf("\nEnter file name:");
 scanf("%d",&x);
 printf("file name is:%d\n",x);
 i=x-1;
 printf("Index is:%d",sb[i]);
```

```
 printf("Block occupied are:");
 for(j=0;j<m[i];j++)
          printf("%3d",b[i][j]);
 getch();
}
```

OUTPUT:

```
Enter no. of files:2
 Enter starting block and size of file1: 2   5
Enter blocks occupied by file1:10
enter blocks of file1:3
2  5   4  6  7   2   6  4  7
Enter starting block and size of file2: 3   4
Enter blocks occupied by file2:5
enter blocks of file2: 2  3   4  5   6
File     index  length
 1       2       10
2       3       5
Enter file name: venkat
file name is:12803
Index is:0Block occupied are:
```

Exp no:2(c) LINKED FILE ALLOCATION

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;
 clrscr();
 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");
 }
 getch();
}
```

OUTPUT:

```
Enter no. of files:2
Enter file name:venkat
Enter starting block:20
Enter no.of blocks:6
Enter block numbers: 4
12
15
45
32
25
Enter file name:rajesh
Enter starting block:12
Enter no.of blocks:5
Enter block numbers:6
5
4
3
2
File    start   size    block
venkat  20      6       4--->12--->15--->45--->32--->25
rajesh  12      5       6--->5--->4--->3--->2
```

Exp. No: 3(a) MULTIPROGRAM VARIABLE TASK

AIM: Write a program to implement Dynamic allocation of memories in MVT.

Algorithm:

Step1: start the process.
Step2: Declare variables.
Step3: Enter total memory size.
Step4: Allocate memory for os.
Step5: allocate total memory to the pages.
Step6: Display the wastage of memory.
Step7: Stop the process.

```
                /* MVT */

#include<stdio.h>
#include<conio.h>
main()
{
        int i,m,n,tot,s[20];
        clrscr();
        printf("Enter total memory size:");
        scanf("%d",&tot);
        printf("Enter no. of pages:");
        scanf("%d",&n);
        printf("Enter memory for OS:");
        scanf("%d",&m);
        for(i=0;i<n;i++)
        {
                printf("Enter size of page%d:",i+1);
                scanf("%d",&s[i]);
        }
        tot=tot-m;
        for(i=0;i<n;i++)
        {
                if(tot>=s[i])
                {
                        printf("Allocate page %d\n",i+1);
                        tot=tot-s[i];
                }
                else
                        printf("process p%d is blocked\n",i+1);
        }
        printf("External Fragmentation is=%d",tot);
        getch();
}
```

OUTPUT:

```
Enter total memory size : 50
Enter no.of pages          : 4
Enter memory for OS    :10

Enter size of page  : 10
Enter size of page  : 9
Enter size of page  : 9
Enter size of page  : 10


External Fragmentation is = 2
```

Exp. No: 3(b) MULTIPROGRAM FIXED TASK

AIM: Write a program to implement Dynamic allocation of memories in MVT.

Algorithm:

Step1: start the process.
Step2: Declare variables.
Step3: Enter total memory size.
Step4: Allocate memory for os.
Step5: allocate total memory to the pages.
Step6: Display the wastage of memory.
Step7: Stop the process.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
        int ms,i,ps[20],n,size,p[20],s,intr=0;
        clrscr();
        printf("Enter size of memory:");
        scanf("%d",&ms);
        printf("Enter memory for OS:");
        scanf("%d",&s);
        ms-=s;
        printf("Enter no.of partitions to be divided:");
        scanf("%d",&n);
        size=ms/n;
        for(i=0;i<n;i++)
        {
                printf("Enter process and process size");
                scanf("%d%d",&p[i],&ps[i]);
                if(ps[i]<=size)
                {
                        intr=intr+size-ps[i];
                        printf("process%d is allocated\n",p[i]);
                }
                else
                        printf("process%d is blocked",p[i]);
        }
        printf("total fragmentation is %d",intr);
        getch();
}
```

OUTPUT:

Enter total memory size : 50

Enter memory for OS :10

Enter no.of partitions to be divided:4

Enter size of page : 10
Enter size of page : 9
Enter size of page : 9
Enter size of page : 8

Internal Fragmentation is = 4

Ex. No: 4 BANKER'S ALGORITHM

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

Deadlock avoidance & Dead Lock Prevention

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");
     getch();
}
```

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

Ex. No: 5(a) FIFO PAGE REPLACEMENT ALGORITHM

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

ALGORITHM:

FIFO:
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()
{
 clrscr();
 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);
 getch();
}
```

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
```

Ex. No:5(b) LRU PAGE REPLACEMENT ALGORITHM

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

LRU (Lease 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()
{
  clrscr();
  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

Ex. No: 5(c) OPTIMAL(LFU) PAGE REPLACEMENT ALGORITHM

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()
{
  clrscr();
  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);
  getch();
}
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:

OPTIMAL PAGE REPLACEMENT ALGORITHM

```
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->        2   3   4
Reference NO 1->        2   3   1
```

No.of page faults...6

Ex. No: 6 PAGING

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();
       //       clrscr();
        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();
       //       clrscr();
        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();
        getch();
}
```

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 theLogical Address: 3

The Physical Address where the instruction present: 1013

Ex. No:7 SEGMENTATION

AIM:

To implement the memory management policy-segmentation.

ALGORITHM:

Step 1: Start the program.
Step 2: Get the number of segments.
Step 3: get the base address and length for each segment.
Step 4: Get the logical address.
Step 5: check whether the segment number is within the limit, if not display the error message.
Step 6: Check whether the byte reference is within the limit, if not display the error message.
Step 7: Calculate the physical memory and display it.
Step 8: Stop the program.

/*MEMORY SEGMENT TABLE*/

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int sost;
void gstinfo();
void ptladdr();

struct segtab
{
        int sno;
        int baddr;
        int limit;
        int val[10];
}st[10];

void gstinfo()
{
        int i,j;
        printf("\n\tEnter the size of the segment table: ");
        scanf("%d",&sost);

        for(i=1;i<=sost;i++)
                {
                printf("\n\tEnter the information about segment: %d",i);
                st[i].sno = i;
                printf("\n\tEnter the base Address: ");
                scanf("%d",&st[i].baddr);
                printf("\n\tEnter the Limit: ");
                scanf("%d",&st[i].limit);
                for(j=0;j<st[i].limit;j++)
                        {
                        printf("Enter the %d address Value: ",(st[i].baddr + j));
                        scanf("%d",&st[i].val[j]);
                        }
                }
}
```

```
void ptladdr()
{
	int i,swd,d=0,n,s,disp,paddr;
	clrscr();
	printf("\n\n\t\t\t SEGMENT TABLE \n\n");
	printf("\n\t	SEG.NO\tBASE ADDRESS\t	LIMIT \n\n");
	for(i=1;i<=sost;i++)
		printf("\t\t%d	\t\t%d\t\t%d\n\n",st[i].sno,st[i].baddr,st[i].limit);
	printf("\n\nEnter the logical Address: ");
	scanf("%d",&swd);
	n=swd;
	while (n != 0)
	{
		n=n/10;
		d++;
	}

	s = swd/pow(10,d-1);
	disp = swd%(int)pow(10,d-1);

	if(s<=sost)
	{
		if(disp < st[s].limit)
		{
			paddr = st[s].baddr + disp;
			printf("\n\t\tLogical Address is: %d",swd);
			printf("\n\t\tMapped Physical address is: %d",paddr);
			printf("\n\tThe value is: %d",( st[s].val[disp] )  );
		}
		else
			printf("\n\t\tLimit of segment %d is high\n\n",s);
	}

	else
		printf("\n\t\tInvalid Segment Address \n");
	}

	void main()
	{
	char ch;
	clrscr();
	gstinfo();
	do
	{
	ptladdr();
```

```
        printf("\n\t Do U want to Continue(Y/N)");
        flushall();
        scanf("%c",&ch);
        }while (ch == 'Y' || ch == 'y' );

        getch();
        }
```

OUTPUT:

```
        Enter the size of the segment table: 3

        Enter the information about segment: 1
        Enter the base Address: 4

        Enter the Limit: 5
Enter the 4 address Value: 11
Enter the 5 address Value: 12
Enter the 6 address Value: 13
Enter the 7 address Value: 14
Enter the 8 address Value: 15

        Enter the information about segment: 2
        Enter the base Address: 5

        Enter the Limit: 4
Enter the 5 address Value: 21
Enter the 6 address Value: 31
Enter the 7 address Value: 41
Enter the 8 address Value: 51

        Enter the information about segment: 3
        Enter the base Address: 3

        Enter the Limit: 4
Enter the 3 address Value: 31
Enter the 4 address Value: 41
Enter the 5 address Value: 41
Enter the 6 address Value: 51

SEGMENT TABLE

            SEG.NO    BASE ADDRESS          LIMIT

              1                  4                    5
```

2	5	4
3	3	4

Enter the logical Address: 3
Logical Address is: 3
Mapped Physical address is: 3
The value is: 31
Do U want to Continue(Y/N)

SEGMENT TABLE

SEG.NO	BASE ADDRESS	LIMIT
1	4	5
2	5	4
3	3	4

Enter the logical Address: 1

Logical Address is: 1
Mapped Physical address is: 4
The value is: 11
Do U want to Continue(Y/N)

Exp No: 1 LEXICAL ANALYZER

Aim:

To write a program for dividing the given input program into lexemes.

ALGORITHM:

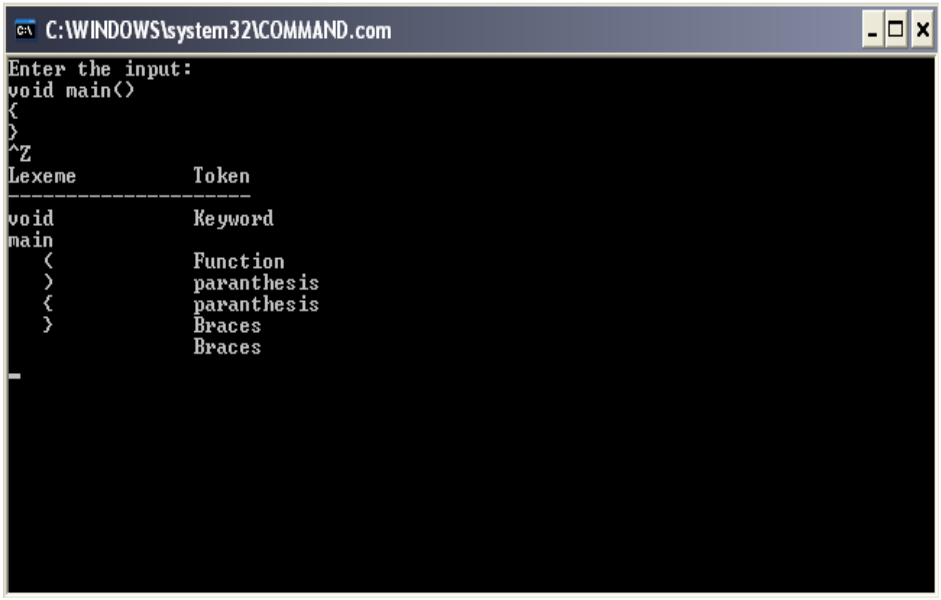
Step 1: start
Step 2: Read the file and open the file as read mode.
Step 3: Read the string for tokens identifiers, variables.
Step 4: take parenthesis also a token.
Step 5: parse the string
Step 6: stop.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
 int i,j,k,p,c;
 char s[120],r[100];
 char par[6]={'(',')','{','}','[',']'};
 char sym[9]={'.',';',':',',','<','>','?','$','#'};
 char key[9][10]={"main","if","else","switch","void","do","while","for","return"};
 char dat[4][10]={"int","float","char","double"};
 char opr[5]={'*','+','-','/','^'};
 FILE *fp;
 clrscr();
 printf("\n\n\t enter the file name");
 scanf("%s",s);
 fp=fopen(s,"r");
 c=0;
 do
 {  fscanf(fp,"%s",r);
  getch();
  for(i=0;i<6;i++)
   if(strchr(r,par[i])!=NULL)
    printf("\n paranthesis :%c",par[i]);
  for(i=0;i<9;i++)
   if(strchr(r,sym[i])!=NULL)
    printf("\n symbol :%c",sym[i]);
  for(i=0;i<9;i++)
   if(strstr(r,key[i])!=NULL)
    printf("\n keyword :%s",key[i]);
  for(i=0;i<4;i++)
   if((strstr(r,dat[i])&&(!strstr(r,"printf")))!=NULL)
    {
     printf("\n data type :%s",dat[i]);
     fscanf(fp,"%s",r);
     printf("\n identifiers :%s",r);
    }
```

```
 for(i=0;i<5;i++)
  if(strchr(r,opr[i])!=NULL)
   printf("\n operator :%c",opr[i]);
  p=c;
  c=ftell(fp);
 }
while(p!=c);
return 0; }
```

Output:



Exp No:2 FIRST FUNCTION

AIM: Write a program to compute FIRST function.

ALGORITHM:

Step 1: Start
Step 2: Declare FILE *fp
Step 3: open the file in.txt in write mode
Step 4: Read the Productions
Step 5: Compute Follow function
Step 6: stop the procuctions.

```
/* FIRST FUNCTION*/

#include<stdio.h>
#include<string.h>
char res[10]={"  "},first[10];
int l=0,count=0,j,term=0;
FILE *fp1;
void main()
{

   FILE *fp;
   int i=0,k=0,n,a[10],set,tem[10]={0},t;
   char ch,s;
   clrscr();
   printf("Enter the productions..\n");
   fp=fopen("input.txt","w");
   while((ch=getchar())!=EOF)
   putc(ch,fp);
     fclose(fp);
     fp=fopen("input.txt","r");
     /*calculation of production starting variable*/
     while(!(feof(fp)))
     {
          ch=fgetc(fp);
          if(feof(fp))
          break;
          first[l++]=ch;
          count++;
          a[i++]=count;
          while(ch!='\n')
          {
            count++;
            ch=fgetc(fp);
          }
          count++;
     }
     rewind(fp);
     n=l;
```

```
        clrscr();
        j=0;
        l=0;
        while(!(feof(fp)))
        {
             ch=fgetc(fp);
             if(feof(fp))
               break;
             while(ch!='\n')
             {
               ch=fgetc(fp);
               if(count==1)
               {

if(((ch>='a')&&(ch<='z'))||ch=='+'||ch=='-'||ch=='*'||ch=='/'||ch=='^'||ch==')'||ch=='('||(
ch=='^')||ch=='#')
                     {
                       if(term!=l||ch!='#')
                          unione(ch,j++);
                       if((term==l)&&(ch=='#'))
                          term=2;
                       count=0;
                      }
                  else
                     {
                      tem[++k]=ftell(fp);
                      set=1;
                     }
                }

               if(ch=='>'||ch=='l')
               {
                  count=1;
               }
               if(set==1)
               {
                     for(i=0;i<n;i++)
                     {
                       fseek(fp,a[i]-1,0);
                       s=fgetc(fp);
                       if(s==ch)
                       {
                          term=1;
                          break;
                       }
                      }
```

```
                    count=0;
                    set=0;
                }
        }
        if(tem[k]!=0)
        {
             t=tem[k]-1;
             tem[k--]=0;
            fseek(fp,t,0);
            if(term==2)
              count=1;
            fgetc(fp);
            ch=fgetc(fp);
           if((k==0&&term==2)&&(ch=='\n'||ch=='l'))
              unione('#',j++);
           fseek(fp,t,0);
          }
          else
            j=print();
    }
   getch();
}
unione(char ch,int j)
{
    int i;
    for(i=0;i<j;i++)
      if(res[i]==ch)
           return;
            res[++j]=ch;

}
print()
{
   static int k=0;
   if(k==0)
   {
       fp1=fopen("output.txt","w");
       k=1;
   }
   printf("first[%c]==",first[l]);
    fputc(first[l++],fp1);
   for(j=0;res[j]!='\0';j++)
   {
      printf("%c",res[j]);
      fputc(res[j],fp1);
   }
```

```
   printf("\n");
   for(j=0;res[j]!='\0';j++)
   res[j]=' ';
   count=0;
   term=0;
   fputc('\n',fp1);
   return 0;
}
```

OUTPUT:

```
Enter the productions
  S->cc
  C->cc
  C->id

First[s]=c
First[c]=id
First[c]=id
```

Exp No:3 FOLLOW FUNCTION

AIM: Write a program to compute FOLLOW(A)

ALGORITHM:

Step 1: Start
Step 2: Declare FILE *fp
Step 3: open the file in.txt in write mode
Step 4: Read the Productions
Step 5: Compute Follow function
Step 6: stop the procuctions.

```
        /*computation  of FOLLOW(A) */

#include<stdio.h>
#include<conio.h>
char ch,first[10],stack[10];
int i,j,k;
void main()
{
   FILE *fp;
   clrscr();
   fp=fopen("in txt","w");
   printf("Enter the productions..\n");
   while(((ch=getchar())!='@'))
   putc(ch,fp);
   fclose(fp);
   fp=fopen("in txt","r");
   i=0;
   while(!(feof(fp)))
   {
         ch=fgetc(fp);
         if(feof(fp))
            break;
         first[i++]=ch;
         while(ch!='\n')
           ch=fgetc(fp);
   }
   rewind(fp);
   i=0;j=0;
   while(first[i]!='\0')
   {
     ch=getc(fp);
     if(ch==first[i])
          stack[j]='$';
     else
           while(!(feof(fp)))
           {
```

```
                while(ch!='>')
                      ch=getc(fp);
                while(ch!=first[i])
                {
                       if(feof(fp))
                          goto down;
                       ch=fgetc(fp);
                }
                ch=fgetc(fp);
                stack[j]=ch;
                down : j++;
              }
              print();
             i++;
       }
    getch();
}
print()
{
     printf("FOLLOW[%c]={",first[i]);
     for(k=0;stack[k]!='\0';k++)
        printf("%c",stack[k]);
    printf("}\n");
    return 0;
}
```

OUTPUT:

```
Enter the productions
  S -> ( l ) / a
  L ->  b

Follow ( s )= { $ }
Follow ( L )= { )  }
```

Exp No:4 OPERATOR PRECEDENCE PARSING

AIM: Write a program to implement operator precedence parsing

ALGORITHM:

Step 1: start.
Step 2: Declare the prototypes for functions.
Step 3: Enter the String like id*id+id
Step 4: Read the string and analyze tokens, identifiers, variables.
Step 5: Display the operator precedence table.
Step 6: stop.

PROGRAM:

```
//OPERATOR PRECEDENCE PARSER
#include<stdio.h>
#include<conio.h>
#include<string.h>
char *p;
e();
t();
main()
{
   int i,j=0,n,b[10],k=0;
   char a[10],c[10];
   clrscr();
   printf("Enter the string\n");
   scanf("%s",a);
   for(i=0,j=0;i<strlen(a);i++)
   {
      switch(a[i])
      {
           case '+' :
           case '-' :
                        c[j]=a[i];
                        b[j]=1;
                        j++;
                        break;
           case '*' :
           case '/' :
                        c[j]=a[i];
                        b[j]=2;
                        j++;
                        break;
           case '^' :
                       c[j]=a[i];
                       b[j]=3;
                       j++;
                       break;
```

```
            default :
                        if(k==0)
                        {
                           k=1;
                           c[j]=a[i];
                           b[j]=4;
                           j++;
                        }
      }
   }
   c[j]='$';
   b[j]=0;
   j++;
   printf("\n\n");
   printf("\n\t---------------------------------");
   printf("\n\n");
   for(i=0;i<j;i++)
     printf("\t%c",c[i]);
   printf("\t");
   for(i=0;i<j;i++)
   {
          printf("\n\t---------------------------------");
          printf("\n\n%c",c[i]);
          for(n=0;n<j;n++)
          {
              if(b[i]<b[n])
                    printf("\t<");
              if(b[i]>b[n])
                    printf("\t>");
              if(b[i]==b[n])
                   printf("\t=");
          }
          printf("\t");
   }
   printf("\n\t-------------------------------------");
   p=a;
   if(e())
      printf("\n\n String parsed");
   else
      printf("\n String not parsed");
   getch();
   return 0;
}
int e()
{
   if(*p=='i')
```

```
   {
      p++;
      t();
      t();
   }
   else
     return 0;
}
int t()
{
   if(*p==NULL)
       return 1;
   else
     if(*p=='+'||*p=='*')
     {
          p++;
          if(*p=='i')
          {
             p++;
          }
          else
          {
             return 0;
          }
     }
     else
       return 0;
}
```

OUTPUT:

```
Enter the string
  I + I * i
---------------------------------
i       +       *       $
---------------------------------
i       =       >       >       >
---------------------------------
+       <       =       <       >
---------------------------------
*       <       >       =       >
  ---------------------------------
$       <       <       <       =
 -------------------------------------
```

String parsed

Exp No:5 RECURSIVE DECENDENT PARSING

AIM: To write a program on recursive descendent parsing.

ALGORITHM:

Step 1: start.
Step 2: Declare the prototype functions E() , EP(),T(), TP(),F()
Step 3: Read the string to be parsed.
Step 4: Check the productions
Step 5: Compare the terminals and Non-terminals
Step 6: Read the parse string.
Step 7: stop the production

PROGRAM:

```
//RECURSIVE DECENT PARSER
#include<stdio.h>
#include<conio.h>
char *p;
E();
EP();
T();
F();
TP();
int main()
{
   clrscr();
   printf("The grammer is\n");
   printf("E->E+T/T\n");
   printf("T->T*F/F\n");
   printf("F->(E)/id\n");
   printf("\n Enter the string to be parsed:");
   scanf("%s",p);
   if(E())
      printf("\nString parsed");
   else
      printf("\nString not parsed");
   getch();
   return 0;
}
E()
{
   T();
   if(EP())
     return 1;
   else
     return 0;
}
EP()
{
```

```
   if(*p=='+')
   {
      p++;
      T();
      EP();
   }
   else if(*p=='\0')
      return 1;
   else
      return 0;

}
T()
{
  F();
  if(TP())
    return 1;
  else
     return 0;
}
TP()
{

   if(*p=='*')
   {
      p++;
      F();
      TP();
   }
    else if(*p=='\0')
      return 0;
}
F()
{
   if(*p=='i')
      p++;
   if(*p=='d')
      p++;
   else if(*p=='(')
     {
      p++;
      F();
      F();
     }
     else if(*p==')')
          p++;
```

```
        else
            return 0;
}
```

OUTPUT:

```
The grammar is
   E -> E+T/T
   T -> T*F/F
    F -> ( E )/id

Enter the string to be parsed id + id * id

String parsed
```