PONDICHERRY UNIVERSITY

(A CENTRAL UNIVERSITY)

SCHOOL OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

MASTER OF COMPUTER APPLICATIONS

CSCA 425: OPERATING SYSTEM LAB

NAME : S. SHALINI

REG. NO. : 21352049

BATCH : 2021-2023

SEMESTER : II SEMESTER

PONDICHERRY UNIVERSITY

(School of Engineering and Technology)

BONAFIDE CERTIFICATE

This is to certify that the CSCA 425 Operating Systems – Lab record is a bonafide record of work done **by S. SHALINI (Reg. No.:21352049)** in partial fulfilment for the award of the degree of Master of Computer Application (MCA), Department of Computer Science, School of Engineering and Technology, Pondicherry University.

FACULTY IN-CHARGE

D r. R. SUNITHA

Assistant Professor,

Department of Computer Science, Science,

HEAD OF THE DEPARTMENT

Dr. S. SIVASATHYA

Professor and Head,

Department of Computer

School of Engineering and Technology, Pondicherry University, University,
Pondicherry – 605014.

School of Engineering and Technology, Pondicherry
Pondicherry – 605014.

Submitted for the practical Examination held on 22/06/22.

INTERNAL EXAMINER

EXTERNAL EXAMINER

Ex.No: 1

FIRST COME FIRST SERVE SCHEDULING (FCFS)

Program Coding:

```
//FCFS
#include <iostream>
#include <algorithm>
#include <iomanip>
using namespace std;
struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};
bool compareArrival(process p1, process p2)
{
```

```
    return p1.arrival_time < p2.arrival_time;
}
bool compareID(process p1, process p2)
{
    return p1.pid < p2.pid;
}
int main() {
    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    cout << setprecision(2) << fixed;
    cout<<"Enter the number of processes: ";
    cin>>n;
   for(int i = 0; i < n; i++) {
    cout<<"Enter arrival time of process "<<i+1<<": ";
```

```
        cin>>p[i].arrival_time;
        cout<<"Enter burst time of process "<<i+1<<": ";
        cin>>p[i].burst_time;
        p[i].pid = i+1;
        cout<<endl;
    }
sort(p,p+n,compareArrival);
for(int i = 0; i < n; i++) {
        p[i].start_time = (i == 0)?p[i].arrival_time:max(p[i-
1].completion_time,p[i].arrival_time);
        p[i].completion_time = p[i].start_time + p[i].burst_time;
        p[i].turnaround_time = p[i].completion_time -
p[i].arrival_time;
        p[i].waiting_time = p[i].turnaround_time -
p[i].burst_time;
        p[i].response_time = p[i].start_time - p[i].arrival_time;


        total_turnaround_time += p[i].turnaround_time;
        total_waiting_time += p[i].waiting_time;
        total_response_time += p[i].response_time;
        total_idle_time += (i ==
0)?(p[i].arrival_time):(p[i].start_time - p[i-
1].completion_time);
    }
```

```
    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cpu_utilisation = ((p[n-1].completion_time - 
total_idle_time) / (float) p[n-1].completion_time)*100;
    throughput = float(n) / (p[n-1].completion_time - 
p[0].arrival_time);


    sort(p,p+n,compareID);


    cout<<endl;

cout<<"#P\t"<<"AT\t"<<"BT\t"<<"CT\t"<<"TAT\t"<<"WT\t
"<<"\n"<<endl;


    for(int i = 0; i < n; i++) {

cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_ti
me<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_tim
e<<"\t"<<p[i].waiting_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = 
"<<avg_turnaround_time<<endl;
```

```
    cout<<"Average Waiting Time =
"<<avg_waiting_time<<endl;
    }
```

OUTPUT:

```
Enter the number of processes: 5
Enter arrival time of process 1: 2
Enter burst time of process 1: 2

Enter arrival time of process 2: 0
Enter burst time of process 2: 1

Enter arrival time of process 3: 2
Enter burst time of process 3: 3

Enter arrival time of process 4: 3
Enter burst time of process 4: 5

Enter arrival time of process 5: 4
Enter burst time of process 5: 4


#P      AT      BT      CT      TAT     WT

1       2       2       4       2       0

2       0       1       1       1       0

3       2       3       7       5       2

4       3       5       12      9       4

5       4       4       16      12      8

Average Turnaround Time = 5.80
Average Waiting Time = 2.80


...Program finished with exit code 0
Press ENTER to exit console.
```

Ex.No: 2

SJF - Preemptive and Non - Preemptive

Program Coding:

SJF Preemptive

```
//SJF Preemptive
#include<stdio.h>
struct process
{
   int WT,AT,BT,TAT;
};

struct process a[10];

int main()
{
   int n,temp[10];
   int count=0,t=0,short_P;
   float total_WT=0, total_TAT=0,Avg_WT,Avg_TAT;
   printf("Enter the number of the process\n");
   scanf("%d",&n);
   printf("Enter the arrival time and burst time of the
process\n");
```

```
    printf("AT BT\n");
    for(int i=0;i<n;i++)
    {
       scanf("%d%d",&a[i].AT,&a[i].BT);

       temp[i]=a[i].BT;
    }

    a[9].BT=10000;
    for(t=0;count!=n;t++)
    {
       short_P=9;
       for(int i=0;i<n;i++)
       {
          if(a[i].BT<a[short_P].BT && (a[i].AT<=t &&
a[i].BT>0))
          {
             short_P=i;
          }

       }

       a[short_P].BT=a[short_P].BT-1;
```

```
    if(a[short_P].BT==0)
    {
        count++;
        a[short_P].WT=t+1-a[short_P].AT-temp[short_P];
        a[short_P].TAT=t+1-a[short_P].AT;

        // total calculation
        total_WT=total_WT+a[short_P].WT;
        total_TAT=total_TAT+a[short_P].TAT;
    }



}
Avg_WT=total_WT/n;
Avg_TAT=total_TAT/n;

printf("Id\tWT\tTAT\n");
for(int i=0;i<n;i++)
{
    printf("%d\t%d\t%d\n",i+1,a[i].WT,a[i].TAT);
}
```

```
    printf("Avg waiting time of the process is %f\n",Avg_WT);
    printf("Avg turn around time of the process
%f\n",Avg_TAT);


}
```

OUTPUT:

```
Enter the number of the process: 5
Enter the arrival time and burst time of the process
AT BT
0  8
1  1
2  3
3  2
4  6
Id      WT      TAT
1       12      20
2       0       1
3       0       3
4       2       4
5       3       9
Avg waiting time of the process is 3.400000
Avg turn around time of the process 7.400000


...Program finished with exit code 0
Press ENTER to exit console.
```

SJF Non-Preemptive

Program Coding:

```
//SJF Non-Preemptive
#include<stdio.h>
 # define max 30
 int main()
 {
    int i,j,n,t,p[max],bt[max],wt[max],tat[max];
    float awt=0,atat=0;
    printf("Enter the number of process");
    scanf("%d",&n);
    printf("Enter the Arrival time. ");
    for(  i = 0; i < n; i++)
    {
      scanf("%d",&p[i]);
    }
    printf("Burst time of process: ");
    for(i=0;i<n;i++)
    {
       scanf("%d",&bt[i]);

    }
```

```
// bubble short
for ( i = 0; i < n; i++)
{
  for(j=0;j<n-i-1;j++)
  {
     if(bt[j]>bt[j+1])
     {
        t=bt[j];
        bt[j]=bt[j+1];
        bt[j+1]=t;

        t=p[j];
        p[j]=p[j+1];
        p[j+1]=t;
     }
  }
}

printf("process\tburst time\t wating time\t turn arount time
\n");
for(i=0;i<n;i++)
{
   wt[i]=0;
```

```
        tat[i]=0;
        for(j=0;j<i;j++)
        {
            wt[i]=wt[i]+bt[j];
        }
        tat[i]=wt[i]+bt[i];
        awt=awt+wt[i];
        atat=atat+tat[i];
        printf("%d\t%d\t\t%d\t\t%d\n",p[i],bt[i],wt[i],tat[i]);

    }
    awt=awt/n;
    atat=atat/n;
    printf("Average wating time =%f\n",awt);
    printf("Average turn around time =%f",atat);



}
```

OUTPUT:

```
Enter the number of process: 5
Enter the Arrival time: 0
1
2
3
4
Burst time of process: 8
4
9
5
6
Process Burst Time       Wating Time     Turn Around Time
1       4               0               4
3       5               4               9
4       6               9               15
0       8               15              23
2       9               23              32
Average wating time =10.200000
Average turn around time =16.600000

...Program finished with exit code 0
Press ENTER to exit console.
```

Ex.No: 3

Round Robin

Program Coding:

```
//Round Robin
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <queue>
#include <cstring>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};
```

```
bool compare1(process p1, process p2)
{
    return p1.arrival_time < p2.arrival_time;
}

bool compare2(process p1, process p2)
{
    return p1.pid < p2.pid;
}

int main() {

    int n;
    int tq;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
```

```
float throughput;
int burst_remaining[100];
int idx;

cout << setprecision(2) << fixed;

cout<<"Enter the number of processes: ";
cin>>n;
cout<<"Enter time quantum: ";
cin>>tq;

for(int i = 0; i < n; i++) {
   cout<<"Enter arrival time of process "<<i+1<<": ";
   cin>>p[i].arrival_time;
   cout<<"Enter burst time of process "<<i+1<<": ";
   cin>>p[i].burst_time;
   burst_remaining[i] = p[i].burst_time;
   p[i].pid = i+1;
   cout<<endl;
}

sort(p,p+n,compare1);
```

```
    queue<int> q;
    int current_time = 0;
    q.push(0);
    int completed = 0;
    int mark[100];
    memset(mark,0,sizeof(mark));
    mark[0] = 1;

    while(completed != n) {
        idx = q.front();
        q.pop();

        if(burst_remaining[idx] == p[idx].burst_time) {
            p[idx].start_time =
max(current_time,p[idx].arrival_time);
            total_idle_time += p[idx].start_time - current_time;
            current_time = p[idx].start_time;
        }

        if(burst_remaining[idx]-tq > 0) {
            burst_remaining[idx] -= tq;
            current_time += tq;
        }
```

```
        else {
            current_time += burst_remaining[idx];
            burst_remaining[idx] = 0;
            completed++;

            p[idx].completion_time = current_time;
            p[idx].turnaround_time = p[idx].completion_time -
p[idx].arrival_time;
            p[idx].waiting_time = p[idx].turnaround_time -
p[idx].burst_time;
            p[idx].response_time = p[idx].start_time -
p[idx].arrival_time;

            total_turnaround_time += p[idx].turnaround_time;
            total_waiting_time += p[idx].waiting_time;
            total_response_time += p[idx].response_time;
        }

        for(int i = 1; i < n; i++) {
            if(burst_remaining[i] > 0 && p[i].arrival_time <=
current_time && mark[i] == 0) {
                q.push(i);
                mark[i] = 1;
            }
```

```
        }
        if(burst_remaining[idx] > 0) {
            q.push(idx);
        }

        if(q.empty()) {
            for(int i = 1; i < n; i++) {
                if(burst_remaining[i] > 0) {
                    q.push(i);
                    mark[i] = 1;
                    break;
                }
            }
        }


    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cpu_utilisation = ((p[n-1].completion_time -
total_idle_time) / (float) p[n-1].completion_time)*100;
```

```
    throughput = float(n) / (p[n-1].completion_time -
p[0].arrival_time);


    sort(p,p+n,compare2);


    cout<<endl;

cout<<"#P\t"<<"AT\t"<<"BT\t"<<"CT\t"<<"TAT\t"<<"WT\t
"<<"\n"<<endl;


    for(int i = 0; i < n; i++) {

cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_ti
me<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_tim
e<<"\t"<<p[i].waiting_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time =
"<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time =
"<<avg_waiting_time<<endl;



}
```

OUTPUT:

```
Enter the number of processes: 4
Enter time quantum: 2
Enter arrival time of process 1: 0
Enter burst time of process 1: 5

Enter arrival time of process 2: 1
Enter burst time of process 2: 4

Enter arrival time of process 3: 2
Enter burst time of process 3: 2

Enter arrival time of process 4: 4
Enter burst time of process 4: 1


#P      AT      BT      CT      TAT     WT

1       0       5       12      12      7

2       1       4       11      10      6

3       2       2       6       4       2

4       4       1       9       5       4

Average Turnaround Time = 7.75
Average Waiting Time = 4.75


...Program finished with exit code 0
Press ENTER to exit console.
```

Ex.No: 4

Priority Preemptive and Non - Preemptive

Priority Preemptive:

Program Coding:

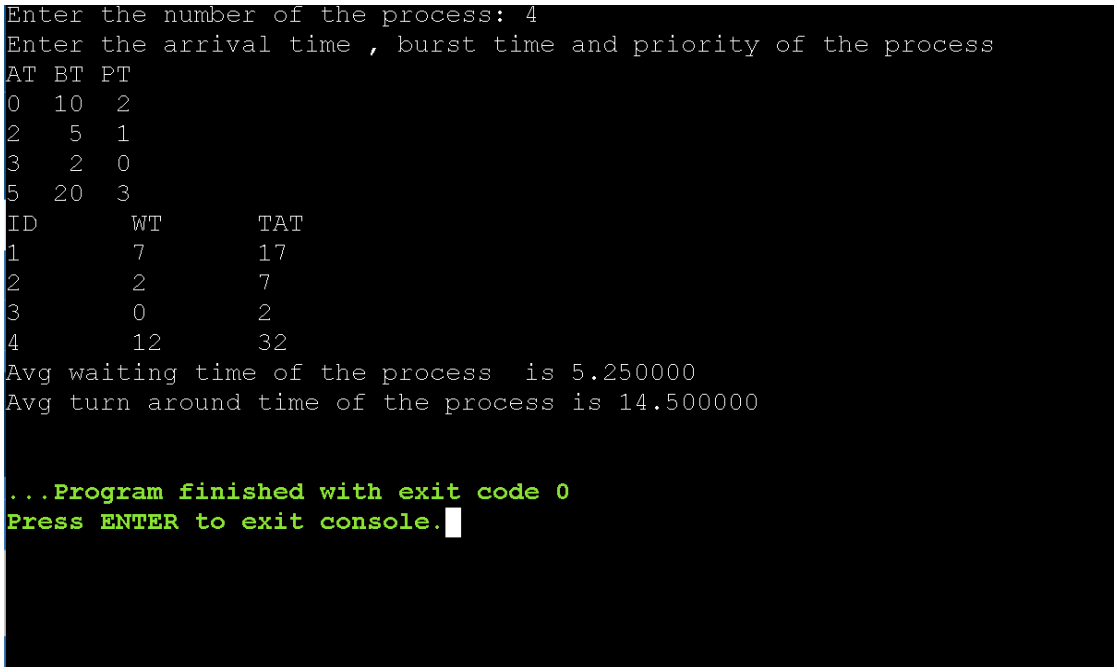
```
//Priority Preemptive
#include<stdio.h>
struct process
{
   int WT,AT,BT,TAT,PT;
};

struct process a[10];

int main()
{
   int n,temp[10],t,count=0,short_p;
   float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
   printf("Enter the number of the process: ");
   scanf("%d",&n);
   printf("Enter the arrival time , burst time and priority of the
process\n");
```

```
    printf("AT BT PT\n");
    for(int i=0;i<n;i++)
    {
       scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);

       temp[i]=a[i].BT;
    }

    a[9].PT=10000;

    for(t=0;count!=n;t++)
    {
       short_p=9;
       for(int i=0;i<n;i++)
       {
          if(a[short_p].PT>a[i].PT && a[i].AT<=t &&
a[i].BT>0)
          {
             short_p=i;
          }
       }

       a[short_p].BT=a[short_p].BT-1;
```

```
    if(a[short_p].BT==0)
    {
        count++;
        a[short_p].WT=t+1-a[short_p].AT-temp[short_p];
        a[short_p].TAT=t+1-a[short_p].AT;

        total_WT=total_WT+a[short_p].WT;
        total_TAT=total_TAT+a[short_p].TAT;

    }
}

Avg_WT=total_WT/n;
Avg_TAT=total_TAT/n;

printf("ID\tWT\tTAT\n");
for(int i=0;i<n;i++)
{
    printf("%d\t%d\t%d\n",i+1,a[i].WT,a[i].TAT);
}
```

```
    printf("Avg waiting time of the process  is
%f\n",Avg_WT);
    printf("Avg turn around time of the process is
%f\n",Avg_TAT);


    return 0;
}
```

OUTPUT:



Priority Non-Preemptive:

Program Coding:

```
//Priority Non-Preemptive
#include<stdio.h>
struct process
{
   int id,WT,AT,BT,TAT,PR;
};
struct process a[10];

void swap(int *b,int *c)
{
   int tem;
   tem=*c;
   *c=*b;
   *b=tem;
}

int main()
{
   int n,check_ar=0;
   int Cmp_time=0;
```

```
    float Total_WT=0,Total_TAT=0,Avg_WT,Avg_TAT;
    printf("Enter the number of process: ");
    scanf("%d",&n);
    printf("Enter the Arrival time , Burst time and priority of
the process\n");
    printf("AT BT PR\n");
    for(int i=0;i<n;i++)
    {
       scanf("%d\t%d\t%d",&a[i].AT,&a[i].BT,&a[i].PR);
       a[i].id=i+1;
       if(i==0)
        check_ar=a[i].AT;

       if(check_ar!=a[i].AT )
        check_ar=1;


    }
    if(check_ar!=0)
    {
       for(int i=0;i<n;i++)
       {
          for(int j=0;j<n-i-1;j++)
          {
```

```
            if(a[j].AT>a[j+1].AT)
            {
                swap(&a[j].id,&a[j+1].id);
                swap(&a[j].AT,&a[j+1].AT);
                swap(&a[j].BT,&a[j+1].BT);
                swap(&a[j].PR,&a[j+1].PR);
            }
        }
    }
}

if(check_ar!=0)
{
    a[0].WT=a[0].AT;
    a[0].TAT=a[0].BT-a[0].AT;
    Cmp_time=a[0].TAT;
    Total_WT=Total_WT+a[0].WT;
    Total_TAT=Total_TAT+a[0].TAT;
    for(int i=1;i<n;i++)
    {
        int min=a[i].PR;
        for(int j=i+1;j<n;j++)
        {
```

```
            if(min>a[j].PR && a[j].AT<=Cmp_time)
            {
                min=a[j].PR;
                swap(&a[i].id,&a[j].id);
                swap(&a[i].AT,&a[j].AT);
                swap(&a[i].BT,&a[j].BT);
                swap(&a[i].PR,&a[j].PR);

            }
        }
          a[i].WT=Cmp_time-a[i].AT;
          Total_WT=Total_WT+a[i].WT;
          Cmp_time=Cmp_time+a[i].BT;

          a[i].TAT=Cmp_time-a[i].AT;
          Total_TAT=Total_TAT+a[i].TAT;

      }
   }   else
   {
      for(int i=0;i<n;i++)
      {
         int min=a[i].PR;
```

```
        for(int j=i+1;j<n;j++)
        {
           if(min>a[j].PR && a[j].AT<=Cmp_time)
           {
              min=a[j].PR;
                swap(&a[i].id,&a[j].id);
                swap(&a[i].AT,&a[j].AT);
                swap(&a[i].BT,&a[j].BT);
                 swap(&a[i].PR,&a[j].PR);
           }

        }
        a[i].WT=Cmp_time-a[i].AT;

        Cmp_time=Cmp_time+a[i].BT;

        a[i].TAT=Cmp_time-a[i].AT;
        Total_WT=Total_WT+a[i].WT;
        Total_TAT=Total_TAT+a[i].TAT;

    }

}
```

```
    Avg_WT=Total_WT/n;
    Avg_TAT=Total_TAT/n;

    printf("The process are\n");
    printf("ID\t WT\t TAT\n");
    for(int i=0;i<n;i++)
    {
       printf("%d\t%d\t%d\n",a[i].id,a[i].WT,a[i].TAT);
    }

    printf("Avg waiting time is: %f\n",Avg_WT);
    printf("Avg turn around time is: %f",Avg_TAT);
    return 0;
}
```

OUTPUT:

```
Enter the number of process: 5
Enter the Arrival time , Burst time and priority of the process
AT BT PR
0  9  5
1  4  3
2  5  1
3  7  2
4  3  4
The process are
ID       WT      TAT
1       0       9
3       7       12
4       11      18
2       20      24
5       21      24
Avg waiting time is: 11.800000
Avg turn around time is: 17.400000

...Program finished with exit code 0
Press ENTER to exit console.
```

Ex.No: 5

Producer Consumer problem using peterson's solution

Program Coding:

```
//Producer Consumer Problem
#include<stdio.h>
#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.Producer\n2.Consumer\n3.Exit");
while(1)
{
printf("\nEnter your choice:");
```

```
scanf("%d",&n);
switch(n)
{
case 1: if((mutex==1)&&(empty!=0))
producer();
else
printf("Buffer is full!!");
break;
case 2: if((mutex==1)&&(full!=0))
consumer();
else
printf("Buffer is empty!!");
break;
case 3:
exit(0);
break;
}
}
return 0;
}

int wait(int s)
{
```

```
return (--s);
}

int signal(int s)
{
return(++s);
}

void producer()
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("\nProducer produces the item %d",x);
mutex=signal(mutex);
}

void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
```

```
printf("\nConsumer consumes item %d",x);
x--;
mutex=signal(mutex);
}
```

OUTPUT:

```
1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
```

```
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:3


...Program finished with exit code 0
Press ENTER to exit console.
```

Ex.No: 6

Dining Philosophers Problem

Program Coding:

```
//Dining Philosophers
#include<stdio.h>
#define n 4
int compltedPhilo = 0,i;
struct fork{
int taken;
}ForkAvil[n];

struct philosp{
int left;
int right;
}Philostatus[n];

void goForDinner(int philID)
{
if(Philostatus[philID].left==10 &&
Philostatus[philID].right==10)
      printf("Philosopher %d completed
hisdinner\n",philID+1);
```

```
else if(Philostatus[philID].left==1 &&
Philostatus[philID].right==1)
{
   printf("Philosopher %d completed his dinner\n",philID+1);
   Philostatus[philID].left = Philostatus[philID].right = 10;
         int otherFork = philID-1;

         if(otherFork== -1)
            otherFork=(n-1);

      ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;
         printf("Philosopher %d released fork %d and fork
%d\n",philID+1,philID+1,otherFork+1);
         compltedPhilo++;
      }
      else if(Philostatus[philID].left==1 &&
Philostatus[philID].right==0)
      {
            if(philID==(n-1))
            {
               if(ForkAvil[philID].taken==0)
               {
                  ForkAvil[philID].taken =
Philostatus[philID].right = 1;
```

```
                printf("Fork %d taken by philosopher
%d\n",philID+1,philID+1);
            }
            else
            {
                printf("Philosopher %d is waiting for fork
%d\n",philID+1,philID+1);
            }
        }else
        {
            int dupphilID = philID;
            philID-=1;

            if(philID== -1)
                philID=(n-1);

            if(ForkAvil[philID].taken == 0){
                ForkAvil[philID].taken =
Philostatus[dupphilID].right = 1;
                printf("Fork %d taken by Philosopher
%d\n",philID+1,dupphilID+1);
            }else{
                printf("Philosopher %d is waiting for Fork
%d\n",dupphilID+1,philID+1);
```

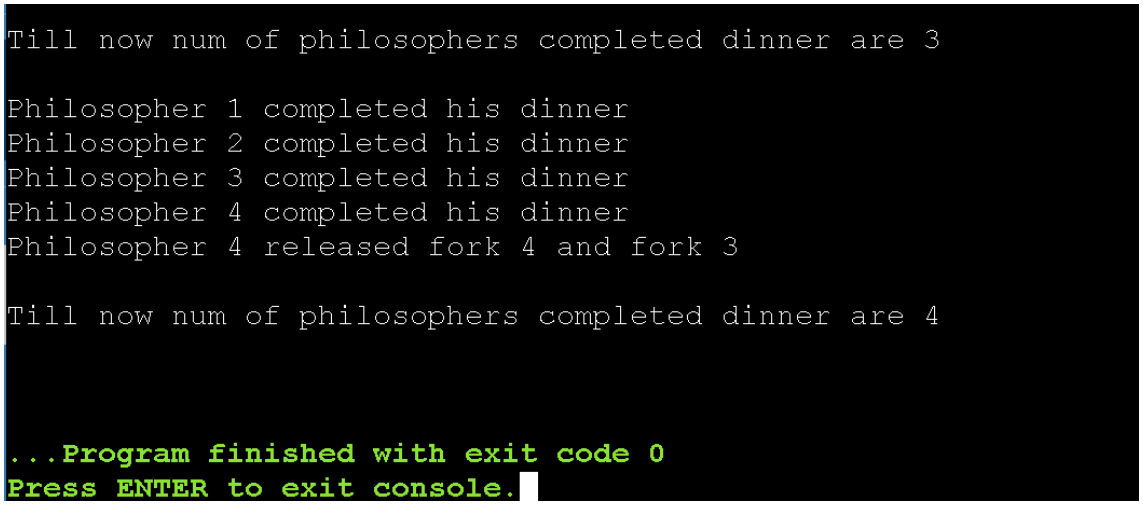
```
            }
        }
    }
    else if(Philostatus[philID].left==0)
    {
        if(philID==(n-1)){
            if(ForkAvil[philID-1].taken==0)
            {
                ForkAvil[philID-1].taken =
Philostatus[philID].left = 1;
                printf("Fork %d taken by philosopher
%d\n",philID,philID+1);
            }else{
                printf("Philosopher %d is waiting for fork
%d\n",philID+1,philID);
            }
        }else
        {
            if(ForkAvil[philID].taken == 0){
                ForkAvil[philID].taken =
Philostatus[philID].left = 1;
                printf("Fork %d taken by Philosopher
%d\n",philID+1,philID+1);
            }else{
```

```
                    printf("Philosopher %d is waiting for Fork
%d\n",philID+1,philID+1);
                }
            }
    }else{}
}

int main(){
for(i=0;i<n;i++)

ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

while(compltedPhilo<n){

for(i=0;i<n;i++)
        goForDinner(i);
printf("\nTill now num of philosophers completed dinner are
%d\n\n",compltedPhilo);
}
 return 0;
}
```

OUTPUT:

```
Fork 1 taken by Philosopher 1
Fork 2 taken by Philosopher 2
Fork 3 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Fork 4 taken by Philosopher 1
Philosopher 2 is waiting for Fork 1
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Philosopher 1 completed his dinner
Philosopher 1 released fork 1 and fork 4
Fork 1 taken by Philosopher 2
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 1

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by Philosopher 3
```

```
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2
Fork 3 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
```


Till now num of philosophers completed dinner are 3
Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
Philosopher 4 released fork 4 and fork 3
Till now num of philosophers completed dinner are 4
...Program finished with exit code 0
Press ENTER to exit console.

Ex.No:7

Readers - Writers Problem

Program Coding:

```
//Reading Writing
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;

void *writer(void *wno)
{
   sem_wait(&wrt);
   cnt = cnt*2;
   printf("Writer %d changed content to %d\n",(*((int
*)wno)),cnt);
   sem_post(&wrt);

}
```

```
void *reader(void *rno)
{
   // Reader acquire the lock before modifying numreader
   pthread_mutex_lock(&mutex);
   numreader++;
   if(numreader == 1) {
      sem_wait(&wrt); // If this id the first reader, then it will
block the writer
   }
   pthread_mutex_unlock(&mutex);
   // Reading Section
   printf("Reader %d: read content as %d\n",*((int *)rno),cnt);

   // Reader acquire the lock before modifying numreader
   pthread_mutex_lock(&mutex);
   numreader--;
   if(numreader == 0) {
      sem_post(&wrt); // If this is the last reader, it will wake
up the writer.
   }
   pthread_mutex_unlock(&mutex);
}
```

```
int main()
{
   pthread_t read[10],write[5];
   pthread_mutex_init(&mutex, NULL);
   sem_init(&wrt,0,1);

   int a[10] = {1,2,3,4,5,6,7,8,9,10}; //Just used for numbering
the producer and consumer

   for(int i = 0; i < 10; i++) {
      pthread_create(&read[i], NULL, (void *)reader, (void
*)&a[i]);
   }
   for(int i = 0; i < 5; i++) {
      pthread_create(&write[i], NULL, (void *)writer, (void
*)&a[i]);
   }

   for(int i = 0; i < 10; i++) {
      pthread_join(read[i], NULL);
   }
   for(int i = 0; i < 5; i++) {
      pthread_join(write[i], NULL);
   }
```

```
    pthread_mutex_destroy(&mutex);
    sem_destroy(&wrt);

    return 0;

}
```

OUTPUT:

```
Reader 1: read content as 1
Reader 2: read content as 1
Reader 5: read content as 1
Reader 6: read content as 1
Reader 4: read content as 1
Reader 3: read content as 1
Reader 9: read content as 1
Writer 1 changed content to 2
Reader 8: read content as 2
Reader 7: read content as 2
Reader 10: read content as 2
Writer 2 changed content to 4
Writer 3 changed content to 8
Writer 5 changed content to 16
Writer 4 changed content to 32


...Program finished with exit code 0
Press ENTER to exit console.
```

Ex.No: 8

File Copy And Move

Program Coding:

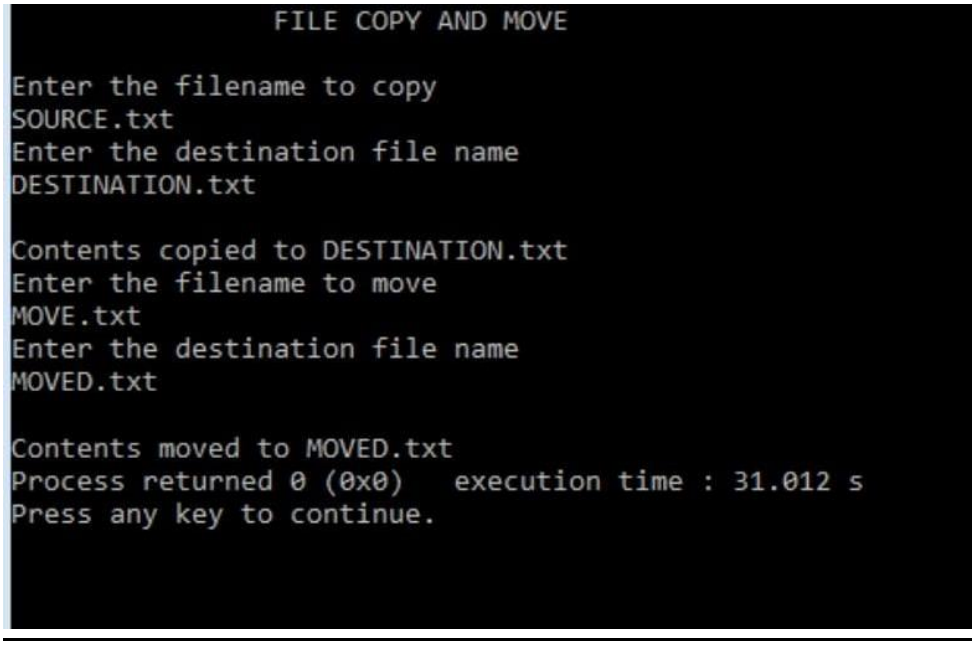
```
//File copy move
#include <stdio.h>
#include <stdlib.h>
void copy_file()
{
 char ch,source_path[100],dest_path[100],dest_filename[100];
 FILE *source_file,*dest_file;
 printf("\nEnter the Source File Path : ");
 scanf("%s",source_path);
 source_file = fopen(source_path,"r");

 if(source_file == NULL)

{
 printf("\nNo such file exists!");

}
 printf("\nEnter the Destination File Path : ");
 scanf("%s",dest_path);
```

```
 dest_file = fopen(dest_path,"w");
 while ((ch = fgetc(source_file)) != EOF)
 fputc(ch, dest_file);
 printf("File copied successfully.\n");
 fclose(source_file);
 fclose(dest_file); }
void move_file() {
 char source_path[100],dest_path[100];
 printf("\nEnter the Source File Path : ");
 scanf("%s",source_path);
 FILE *file;
 file = fopen(source_path,"r");
 if(file == NULL)

{
 printf("\nNo such file exists!");
 return;

}
 fclose(file);
 printf("\nEnter the Destination File Path : ");
 scanf("%s",dest_path);
 if(rename(source_path,dest_path)==0)
```

```
{
 printf("File Moved Successfully\n");
 }
 else

{
 perror(NULL);
 printf("Error occurred while moving file\n");

}
}
int main() {
 int ch;
 printf("\nFILE COPY AND MOVE\n");
 while(1)
 {
 printf("\nChoose one of the option");
 printf("\n1. Copy\n2. Move\n3. Exit\n");
 printf("Enter any choice : ");
 scanf("%d",&ch);
 switch(ch)
 {
```

```
case 1:
copy_file();
break;
case 2:
move_file();
break;
case 3:
exit(1);
break;
default:
printf("The entered choice is incorrect");
}
}
return 0;
}
```

OUTPUT:


FILE COPY AND MOVE
Enter the filename to copy
SOURCE.txt
Enter the destination file name
DESTINATION.txt
Contents copied to DESTINATION.txt
Enter the filename to move
MOVE.txt
Enter the destination file name
MOVED.txt
Contents moved to MOVED.txt
Process returned 0 (0x0) execution time : 31.012 s
Press any key to continue.

Ex.No: 9

File Permissions

Program Coding:

```
//File Permission
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void creating()
{
char file_name[100];
FILE *file;
printf("\nEnter File Name : ");
scanf("%s", file_name);
file = fopen(file_name, "w");
if (file != NULL)
printf("File Created Successfully\n");
else
printf("\nError occured \n");
fclose(file);
}
void writing()
{
```

```
char file_name[100], ch[100];
FILE *file;
printf("\nEnter File Name :");
scanf("%s", file_name);
file = fopen(file_name, "w");
printf("\nEnter the content for file : ");
scanf("%s", ch);
fprintf(file, ch);
printf("data writed to %s Successfully \n", file_name);
fclose(file);
}
void reading()

{
char file_name[100], ch[100];
FILE *file;
printf("\nEnter File Name :");
scanf("%s", file_name);
file = fopen(file_name, "r");
fscanf(file, "%s", ch);
printf("The Content of %s is: %s ", file_name, ch);
fclose(file);
}
```

```
void appending()
{
char file_name[100], ch[100];
FILE *file;
printf("\nEnter File Name :");
scanf("%s", file_name);
file = fopen(file_name, "a");
printf("\nEnter the content for file : ");
do
{
scanf("%s", ch);
fprintf(file, ch);
}
while(!strcmp(ch,"END"));
printf("%s Appended Successfully in %s\n", ch, file_name);
fclose(file);
}
void delet()
{
char file_name[100];
printf("\nEnter File Name :");
scanf("%s", file_name);
if (remove(file_name) == 0)
```

```
printf("Deleted successfully\n");
else
printf("%s not found\n", file_name);
}

int main()
{
int ch;
printf("\n---- FILE OPERATIONS ----\n");
do
{
printf("\nMenu\n");
printf("1: Create file \n");
printf("2: Write to file \n");
printf("3: Read from file \n");
printf("4: Append to file \n");
printf("5: Delete file \n");
printf("6: Exit \n");
printf("Enter your Choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:creating();break;
```

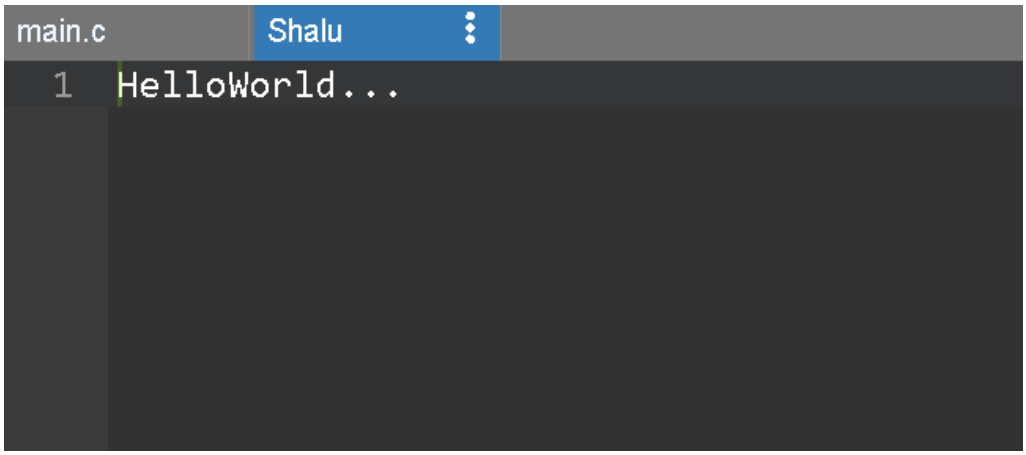
```
case 2:writing();break;
case 3:reading();break;
case 4:appending();break;
case 5:delet();break;
case 6:
printf("Exiting from program...");
exit(1);
default:
printf("Invalid choice\n");
}
}
while(ch!=0);
return 0;
}
```

OUTPUT:

```
---- FILE OPERATIONS ----

Menu
1: Create file
2: Write to file
3: Read from file
4: Append to file
5: Delete file
6: Exit
Enter your Choice:1

Enter File Name : Shalu
File Created Successfully

Menu
1: Create file
2: Write to file
3: Read from file
4: Append to file
5: Delete file
6: Exit
Enter your Choice:2

Enter File Name :Shalu

Enter the content for file : Hello
data writed to Shalu Successfully

Menu
1: Create file
```

```
2: Write to file
3: Read from file
4: Append to file
5: Delete file
6: Exit
Enter your Choice:3

Enter File Name :Shalu
The Content of Shalu is: Hello
Menu
1: Create file
2: Write to file
3: Read from file
4: Append to file
5: Delete file
6: Exit
Enter your Choice:4

Enter File Name :Shalu

Enter the content for file : World...
World... Appended Successfully in Shalu

Menu
1: Create file
2: Write to file
3: Read from file
4: Append to file
5: Delete file
6: Exit
Enter your Choice:3
```

```
Enter File Name :Shalu

Enter the content for file : World...
World... Appended Successfully in Shalu

Menu
1: Create file
2: Write to file
3: Read from file
4: Append to file
5: Delete file
6: Exit
Enter your Choice:3

Enter File Name :Shalu
The Content of Shalu is: HelloWorld...
Menu
1: Create file
2: Write to file
3: Read from file
4: Append to file
5: Delete file
6: Exit
Enter your Choice:6
Exiting from program...

...Program finished with exit code 1
Press ENTER to exit console.
```



Ex.No: 10

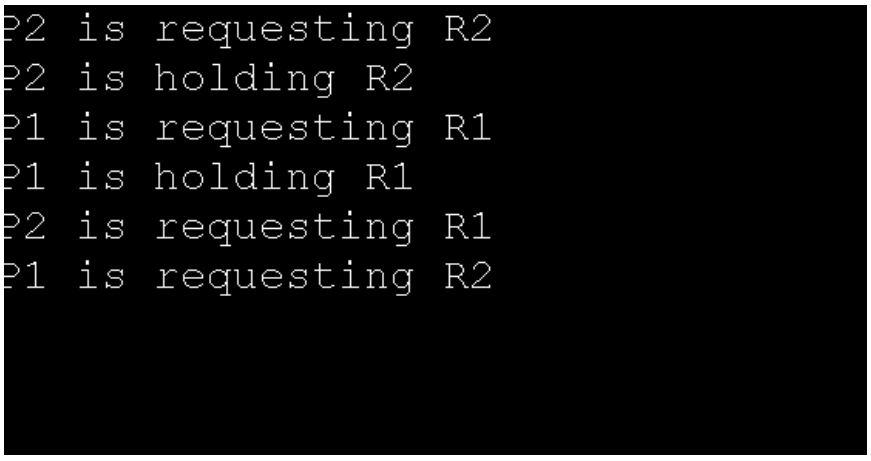
Deadlocks

Program Coding:

```
//Deadlock
#include<stdio.h>
#include<pthread.h>
#include<unistd.h>
#include<stdlib.h>
pthread_mutex_t R1,R2;
void *fun1()
{
  printf("P1 is requesting R1\n");
  pthread_mutex_lock(&R1);
  printf("P1 is holding R1\n");
  sleep(1);
  printf("P1 is requesting R2\n");
  pthread_mutex_lock(&R2);
  printf("P1 is holding R2\n");
  pthread_mutex_unlock(&R1);
  pthread_mutex_unlock(&R2);
}
 void *fun2()
```

```
{
 printf("P2 is requesting R2\n");
 pthread_mutex_lock(&R2);
 printf("P2 is holding R2\n");
 sleep(1);
 printf("P2 is requesting R1\n");
 pthread_mutex_lock(&R1);
 printf("P2 is holding R1\n");
 pthread_mutex_unlock(&R1);
 pthread_mutex_unlock(&R2);
}
int main()
{
 pthread_t p1,p2;
 pthread_mutex_init(&R1,NULL);
 pthread_mutex_init(&R2,NULL);
 pthread_create(&p1,NULL,&fun1,NULL);
 pthread_create(&p2,NULL,&fun2,NULL);
 pthread_join(p1,NULL);
 pthread_join(p2,NULL);
 return 0;
}
```

OUTPUT:


P2 is requesting R2
P2 is holding R2
P1 is requesting R1
P1 is holding R1
P2 is requesting R1
P1 is requesting R2

Ex.No: 11

Banker's Algorithm

Program Coding:

```
//Banker's Algorithm
#include<iostream>
using namespace std;
const int P = 5;
const int R = 3;
void calculateNeed(int need[P][R], int maxm[P][R],int
allot[P][R])
{
      for (int i = 0 ; i < P ; i++)
            for (int j = 0 ; j < R ; j++)
                  need[i][j] = maxm[i][j] - allot[i][j];
}
bool isSafe(int processes[], int avail[], int maxm[][R],int
allot[][R])
{
      int need[P][R];
   calculateNeed(need, maxm, allot);
   bool finish[P] = {0};
   int safeSeq[P];
   int work[R];
```

```
for (int i = 0; i < R ; i++)
        work[i] = avail[i];
int count = 0;
while (count < P)
{
        bool found = false;
        for (int p = 0; p < P; p++)
        {
                if (finish[p] == 0)
                {
                        int j;
                        for (j = 0; j < R; j++)
                                if (need[p][j] > work[j])
                                        break;
                        if (j == R)
                        {
                                for (int k = 0 ; k < R ; k++)
                                        work[k] += allot[p][k];

                                safeSeq[count++] = p;
                                finish[p] = 1;
                                found = true;
                        }
```

```
                }
            }

            if (found == false)
            {
                cout << "System is not in safe state";
                return false;
            }
        }

        cout << "System is in safe state.\nSafe" " sequence is: ";
        for (int i = 0; i < P ; i++)
            cout << safeSeq[i] << " ";

        return true;
}

int main()
{
        int processes[] = {0, 1, 2, 3, 4};
        int avail[] = {3, 3, 2};
        int maxm[][R] = {{7, 5, 3},
                         {3, 2, 2},
```

```
                                {9, 0, 2},
                                {2, 2, 2},
                                {4, 3, 3}};

        int allot[][R] = {{0, 1, 0},
                                {2, 0, 0},
                                {3, 0, 2},
                                {2, 1, 1},
                                {0, 0, 2}};

        isSafe(processes, avail, maxm, allot);
    return 0;
}
```

OUTPUT:

Ex.No: 12

Disk Scheduling Algorithms

a)FCFS

Program Coding:

```
//Disk schedular
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main()
{
   int i,n,req[50],mov=0,cp;
   printf("Enter the Current Position of Head: ");
   scanf("%d",&cp);
   printf("Enter the No. of Requests: ");
   scanf("%d",&n);
   printf("Enter the Requests in Order: ");
   for(i=0;i<n;i++)
   {
      scanf("%d",&req[i]);
   }
```

```
    mov=mov+abs(cp-req[0]);
    printf("%d -> %d",cp,req[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(req[i]-req[i-1]);
        printf(" -> %d",req[i]);
    }
    printf("\n");
    printf("Total Head Movements = %d\n",mov);
}
```

OUTPUT:

```
Enter the Current Position of Head: 55
Enter the No. of Requests: 7
Enter the Requests in Order: 93  176  42  148  27  14  180
55 -> 93 -> 176 -> 42 -> 148 -> 27 -> 14 -> 180
Total Head Movements = 661


...Program finished with exit code 0
Press ENTER to exit console.
```

b) SSTF

Program Coding:

```
//Disk scheduling SSTF
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);

   // logic for sstf disk scheduling

      /* loop will execute until all process is completed*/
   while(count!=n)
   {
```

```
        int min=1000,d,index;
        for(i=0;i<n;i++)
        {
          d=abs(RQ[i]-initial);
          if(min>d)
          {
             min=d;
             index=i;
          }

        }
        TotalHeadMoment=TotalHeadMoment+min;
        initial=RQ[index];
        // 1000 is for max
        // you can use any number
        RQ[index]=1000;
        count++;
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

OUTPUT:

```
Enter the number of Requests
6
Enter the Requests sequence
98 76 54 32 14 51
Enter initial head position
13
Total head movement is 85

...Program finished with exit code 0
Press ENTER to exit console.
```

c) SCAN

Program Coding:

```
//Disk scheduling SCAN
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and
for low 0\n");
   scanf("%d",&move);


   // logic for Scan disk scheduling
```

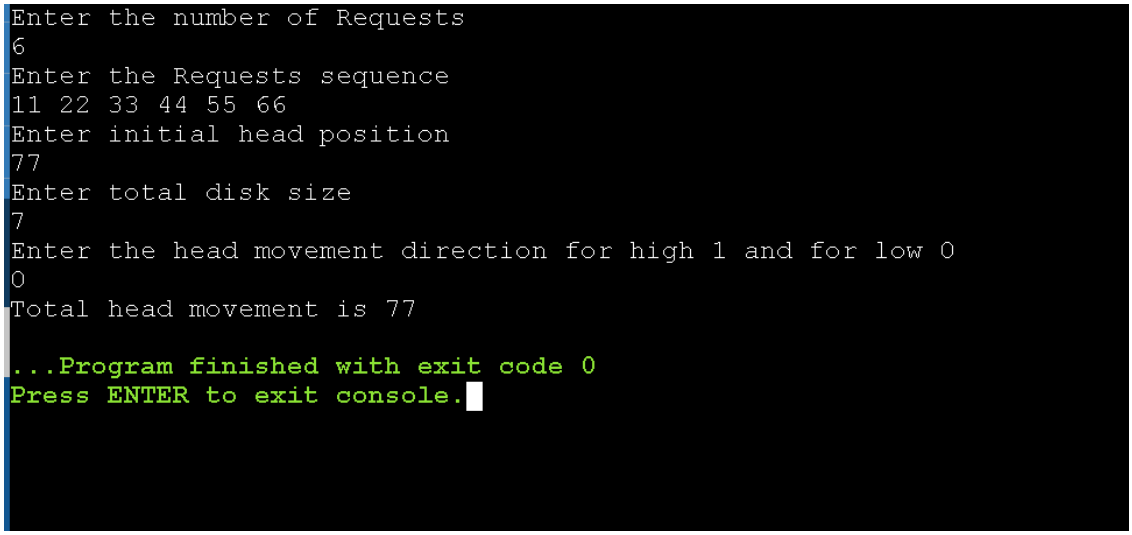
```
    /*logic for sort the request array */
for(i=0;i<n;i++)
{
    for(j=0;j<n-i-1;j++)
    {
        if(RQ[j]>RQ[j+1])
        {
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
        }

    }
}

int index;
for(i=0;i<n;i++)
{
    if(initial<RQ[i])
    {
        index=i;
        break;
```

```
        }
    }

    // if movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
            initial=RQ[i];
        }
        //  last movement for max size
        TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-
1]-1);
        initial = size-1;
        for(i=index-1;i>=0;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
             initial=RQ[i];


        }
    }
```

```
    // if movement is towards low value
    else
    {
       for(i=index-1;i>=0;i--)
       {
          TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
          initial=RQ[i];
       }
       //  last movement for min size
       TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
       initial =0;
       for(i=index;i<n;i++)
       {
          TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
          initial=RQ[i];

       }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
```

}

OUTPUT:


Enter the number of Requests
6
Enter the Requests sequence
11 22 33 44 55 66
Enter initial head position
77
Enter total disk size
7
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 77
...Program finished with exit code 0
Press ENTER to exit console.

d) C – SCAN

Program Coding:

```
//Disk Management CSCAN
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and
for low 0\n");
   scanf("%d",&move);


   // logic for C-Scan disk scheduling
```

```
    /*logic for sort the request array */
for(i=0;i<n;i++)
{
    for( j=0;j<n-i-1;j++)
    {
        if(RQ[j]>RQ[j+1])
        {
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
        }

    }
}

int index;
for(i=0;i<n;i++)
{
    if(initial<RQ[i])
    {
        index=i;
        break;
```

```
        }
    }

    // if movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
            initial=RQ[i];
        }
        //  last movement for max size
        TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-
1]-1);
        /*movement max to min disk */
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial=0;
        for( i=0;i<index;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
            initial=RQ[i];
```

```
        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
            initial=RQ[i];
        }
        //  last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        /*movement min to max disk */
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial =size-1;
        for(i=n-1;i>=index;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
            initial=RQ[i];


        }
```

```
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

OUTPUT:

```
Enter the number of Requests
5
Enter the Requests sequence
98 78 65 56 44
Enter initial head position
19
Enter total disk size
7
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 177

...Program finished with exit code 0
Press ENTER to exit console.
```

e) LOOK

Program Coding:

```
//Disk scheduling LOOK
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and
for low 0\n");
   scanf("%d",&move);


   // logic for look disk scheduling
```

```
   /*logic for sort the request array */
for(i=0;i<n;i++)
{
   for(j=0;j<n-i-1;j++)
   {
      if(RQ[j]>RQ[j+1])
      {
         int temp;
         temp=RQ[j];
         RQ[j]=RQ[j+1];
         RQ[j+1]=temp;
      }

   }
}

int index;
for(i=0;i<n;i++)
{
   if(initial<RQ[i])
   {
      index=i;
      break;
```

```
        }
    }

    // if movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
            initial=RQ[i];
        }

        for(i=index-1;i>=0;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
             initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
```

```
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
            initial=RQ[i];
        }

        for(i=index;i<n;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
             initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

OUTPUT:

```
Enter the number of Requests
4
Enter the Requests sequence
21 76 90 33
Enter initial head position
22
Enter total disk size
6
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 70

...Program finished with exit code 0
Press ENTER to exit console.
```

f) C – LOOK

Program Coding:

```
//Disk scheduling CLOOK
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and
for low 0\n");
   scanf("%d",&move);

   // logic for C-look disk scheduling
```

```
    /*logic for sort the request array */
for(i=0;i<n;i++)
{
    for( j=0;j<n-i-1;j++)
    {
        if(RQ[j]>RQ[j+1])
        {
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
        }

    }
}

int index;
for(i=0;i<n;i++)
{
    if(initial<RQ[i])
    {
        index=i;
        break;
```

```
        }
    }

    // if movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
            initial=RQ[i];
        }

        for( i=0;i<index;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
             initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
```

```
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
            initial=RQ[i];
        }

        for(i=n-1;i>=index;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-
initial);
             initial=RQ[i];

        }
    }

    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```

OUTPUT:

```
Enter the number of Requests
7
Enter the Requests sequence
12 34 45 56 67 78 89
Enter initial head position
11
Enter total disk size
9
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 78

...Program finished with exit code 0
Press ENTER to exit console.
```

Ex.No: 13

Page Replacement Algorithms

FIFO

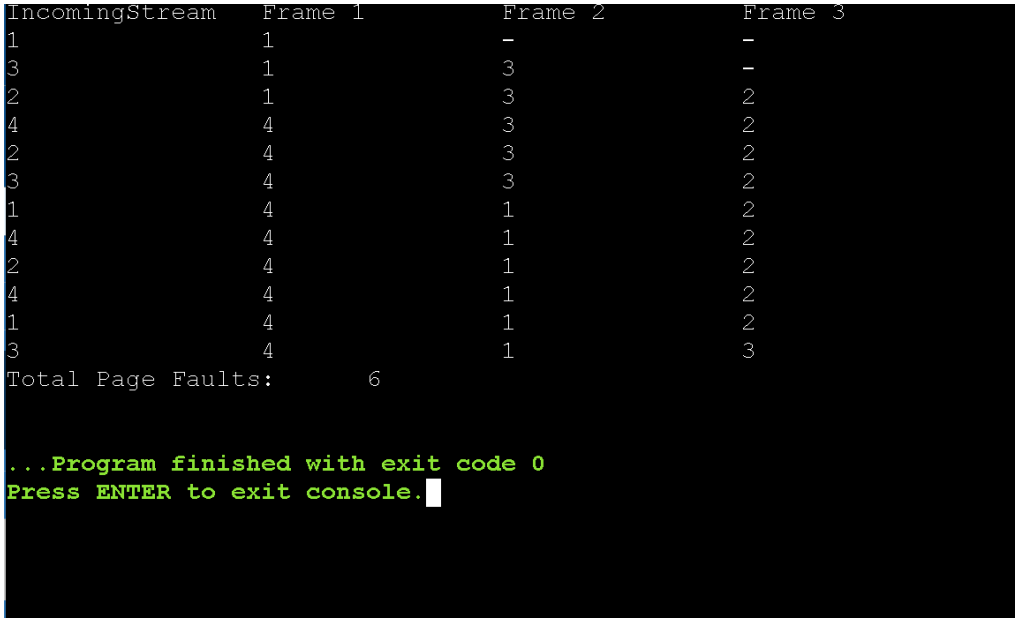
Program Coding:

```
//Page Replacement
#include <stdio.h>
int main()
{
   int incomingStream[] = {1,3,2,4,2,3,1,4,2,4,1,3};
   int pageFaults = 0;
   int frames = 3;
   int m, n, s, pages;

   pages =
sizeof(incomingStream)/sizeof(incomingStream[0]);

   printf("IncomingStream \t Frame 1 \t Frame 2 \t Frame 3");
   int temp[frames];
   for(m = 0; m < frames; m++)
   {
      temp[m] = -1;
   }
```

```
    for(m = 0; m < pages; m++)
    {
        s = 0;

        for(n = 0; n < frames; n++)
        {
            if(incomingStream[m] == temp[n])
            {
                s++;
                pageFaults--;
            }
        }
        pageFaults++;

        if((pageFaults <= frames) && (s == 0))
        {
            temp[m] = incomingStream[m];
        }
        else if(s == 0)
        {
            temp[(pageFaults - 1) % frames] =
incomingStream[m];
```

```
        }

        printf("\n");
        printf("%d\t\t",incomingStream[m]);
        for(n = 0; n < frames; n++)
        {
            if(temp[n] != -1)
                printf(" %d\t\t", temp[n]);
            else
                printf(" - \t\t");
        }
    }

    printf("\nTotal Page Faults:\t%d\n", pageFaults);
    return 0;
}
```

OUTPUT:


IncomingStream Frame 1 Frame 2 Frame 3
1 1 - -
3 1 3 -
2 1 3 2
4 4 3 2
2 4 3 2
3 4 3 2
1 4 1 2
4 4 1 2
2 4 1 2
4 4 1 2
1 4 1 2
3 4 1 3
Total Page Faults: 6
...Program finished with exit code 0
Press ENTER to exit console.

Ex.No: 14

Memory Management

a)Best Fit

Program Coding:

```
//Memory Management BestFit
#include<iostream>
#include<algorithm>
using namespace std;

struct node{
    int memsize;
    int allocp=-1;
    int pos;
    int allocSize;
}m[200];
bool posSort(node a,node b){
    return a.pos < b.pos;
}
bool memSort(node a,node b){
    return a.memsize < b.memsize;
}
int main(){
```

```
int nm,np,choice, i, j, p[200];
cout<<"Enter number of blocks\n";
cin>>nm;
cout<<"Enter block size\n";
for(i=0;i<nm;i++){
    cin>>m[i].memsize;
    m[i].pos=i;
}

cout<<"Enter number of processes\n";
cin>>np;

cout<<"Enter process size\n";
for(i=0;i<np;i++){
    cin>>p[i];
}
cout<<"\n\n";
sort(m,m+nm,memSort);
int globalFlag=0;

for(i=0;i<np;i++){
    int flag=0;
    for(j=0;j<nm;j++){
```

```
        if(p[i]<=m[j].memsize && m[j].allocp==-1){
            m[j].allocp=i;
            m[j].allocSize=p[i];
            flag=1;
            break;
        }
    }
    if(flag==0){
            cout<<"Unallocated Process P"<<i+1<<"\n";
            globalFlag=1;
        }
    }
sort(m,m+nm,posSort);
cout<<"\n";
int intFrag=0,extFrag=0;
cout<<"Memory\t\t";
for(i=0;i<nm;i++){
    cout<<m[i].memsize<<"\t";
}
cout<<"\n";
cout<<"P. Alloc.\t";
for(i=0;i<nm;i++){
    if(m[i].allocp!=-1){
```

```
            cout<<"P"<<m[i].allocp+1<<"\t";
        }
        else{
            cout<<"Empty\t";
        }
    }
    cout<<"\n";
    cout<<"Int. Frag.\t";
    for(i=0;i<nm;i++){
            if(m[i].allocp!=-1){
                cout<<m[i].memsize-m[i].allocSize<<"\t";
                intFrag+=m[i].memsize-m[i].allocSize;
            }
            else{
                extFrag+=m[i].memsize;
                cout<<"Empty\t";
            }
    }
    cout<<"\n";
    cout<<"\n";

    if(globalFlag==1)
```

```
        cout<<"Total External Fragmentation:
"<<extFrag<<"\n";
    else
    {
        cout<<"Available Memory: "<<extFrag<<"\n";
    }

    cout<<"Total Internal Fragmentation: "<<intFrag<<"\n";
    return 0;
}
```

OUTPUT:

```
Enter number of blocks
5
Enter block size
500 400 300 200 100
Enter number of processes
4
Enter process size
90 200 280 300



Memory          500     400     300     200     100
P. Alloc.       Empty   P4      P3      P2      P1
Int. Frag.      Empty   100     20      0       10

Available Memory: 500
Total Internal Fragmentation: 130


...Program finished with exit code 0
Press ENTER to exit console.
```

b) Worst Fit

Program Coding:

```
//Memory management Worst fit
#include<iostream>
#include<algorithm>
using namespace std;

struct node{
    int memsize;
    int allocp=-1;
    int pos;
    int allocSize;
}m[200];


bool posSort(node a,node b){
    return a.pos < b.pos;
}

bool memSort(node a,node b){
    return a.memsize > b.memsize;
}
```

```
int main(){
    int nm,np,choice, i, j, p[200];
    cout<<"Enter number of blocks\n";
    cin>>nm;
    cout<<"Enter block size\n";
    for(i=0;i<nm;i++){
        cin>>m[i].memsize;
        m[i].pos=i;
    }
    cout<<"Enter number of processes\n";
    cin>>np;
    cout<<"Enter process size\n";
    for(i=0;i<np;i++){
        cin>>p[i];
    }
    cout<<"\n\n";
    sort(m,m+nm,memSort);
    int globalFlag=0;
    for(i=0;i<np;i++){
        int flag=0;
        for(j=0;j<nm;j++){
            if(p[i]<=m[j].memsize && m[j].allocp==-1){
```

```
            m[j].allocp=i;
            m[j].allocSize=p[i];
            flag=1;
            break;
        }
    }
    if(flag==0){
            cout<<"Unallocated Process P"<<i+1<<"\n";
            globalFlag=1;
        }
    }
sort(m,m+nm,posSort);
cout<<"\n";
int intFrag=0,extFrag=0;
cout<<"Memory\t\t";
for(i=0;i<nm;i++){
    cout<<m[i].memsize<<"\t";
}
cout<<"\n";
cout<<"P. Alloc.\t";
for(i=0;i<nm;i++){
    if(m[i].allocp!=-1){
        cout<<"P"<<m[i].allocp+1<<"\t";
```

```
        }
        else{
            cout<<"Empty\t";
        }
    }
    cout<<"\n";
    cout<<"Int. Frag.\t";
    for(i=0;i<nm;i++){
        if(m[i].allocp!=-1){
            cout<<m[i].memsize-m[i].allocSize<<"\t";
            intFrag+=m[i].memsize-m[i].allocSize;
        }
        else{
            extFrag+=m[i].memsize;
            cout<<"Empty\t";
        }
    }
    cout<<"\n";
    cout<<"\n";

    if(globalFlag==1)
        cout<<"Total External Fragmentation:
"<<extFrag<<"\n";
```

```
    else{
        cout<<"Available Memory: "<<extFrag<<"\n";
    }
    cout<<"Total Internal Fragmentation: "<<intFrag<<"\n";
    return 0;
}
```

OUTPUT:

```
Enter number of blocks
5
Enter block size
500 400 300 200 100
Enter number of processes
4
Enter process size
90 200 280 300


Unallocated Process P4

Memory          500     400     300     200     100
P. Alloc.       P1      P2      P3      Empty   Empty
Int. Frag.      410     200     20      Empty   Empty

Total External Fragmentation: 300
Total Internal Fragmentation: 630


...Program finished with exit code 0
Press ENTER to exit console.
```

c) First Fit

Program Coding:

```
//Memory management First Fit
#include<iostream>
#include<algorithm>
using namespace std;

struct node{
    int memsize;
    int allocp=-1;
    int pos;
    int allocSize;
}m[200];


bool posSort(node a,node b){
    return a.pos < b.pos;
}

bool memSort(node a,node b){
    return a.memsize < b.memsize;
}
```

```
int main(){
    int nm,np,choice, i, j, p[200];
    cout<<"Enter number of blocks: ";
    cin>>nm;
    cout<<"Enter block size: ";
    for(i=0;i<nm;i++){
        cin>>m[i].memsize;
        m[i].pos=i;
    }
    cout<<"Enter number of processes: ";
    cin>>np;
    cout<<"Enter process size\n";
    for(i=0;i<np;i++){
        cin>>p[i];
    }
    cout<<"\n\n";
    //sort(m,m+nm,memSort);
    int globalFlag=0;

    for(i=0;i<np;i++){
        int flag=0;
        for(j=0;j<nm;j++){
```

```
        if(p[i]<=m[j].memsize && m[j].allocp==-1){
            m[j].allocp=i;
            m[j].allocSize=p[i];
            flag=1;
            break;
        }
    }
    if(flag==0){
            cout<<"Unallocated Process P"<<i+1<<"\n";
            globalFlag=1;
        }
    }
sort(m,m+nm,posSort);
cout<<"\n";
int intFrag=0,extFrag=0;
cout<<"Memory\t\t";
for(i=0;i<nm;i++){
    cout<<m[i].memsize<<"\t";
}
cout<<"\n";
cout<<"P. Alloc.\t";
for(i=0;i<nm;i++){
    if(m[i].allocp!=-1){
```

```
            cout<<"P"<<m[i].allocp+1<<"\t";
        }
        else{
            cout<<"Empty\t";
        }
    }
    cout<<"\n";
    cout<<"Int. Frag.\t";
    for(i=0;i<nm;i++){
        if(m[i].allocp!=-1){
            cout<<m[i].memsize-m[i].allocSize<<"\t";
            intFrag+=m[i].memsize-m[i].allocSize;
        }
        else{
            extFrag+=m[i].memsize;
            cout<<"Empty\t";
        }
    }
    cout<<"\n";
    cout<<"\n";
    if(globalFlag==1)
        cout<<"Total External Fragmentation:
"<<extFrag<<"\n";
```

```
    else
    {
        cout<<"Available Memory: "<<extFrag<<"\n";
    }
        cout<<"Total Internal Fragmentation: "<<intFrag<<"\n";
    return 0;
}
```

OUTPUT: