S.No: 2	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date:

Aim:

Implementation of the Round Robin cpu scheduling algorithm
(https://gecgudlavalleru.codetantra.com/secure/labs-q.jsp?sNo=4&qId=5bec179564bac110545ba035&bd=AY3RFZHVEQg%3D%3D&lid=5db6d168a183970b79e5cd34&labbd=AMzM2X2N0X2No&expTitle=Implementation%20of%20the%20Roun

Source Code:

os4.c

```
#include<stdio.h>
#include<conio.h>
#define max 30
int main()
{
   int i,limit,total=0,x,counter=0,tq;
   int wt=0,tat=0,at[10],bt[10],temp[10];
   float awt,atat;
   printf("Enter Total Number of Processes: ");
   scanf("%d",&limit);
   x=limit;
   for(i=0;i<limit;i++)
   {
      printf("Enter Details of Process[%d]: ",i+1);
      printf("Arrival Time:\t");
      scanf("%d",&at[i]);
      printf("Burst Time:\t");
      scanf("%d",&bt[i]);
      temp[i]=bt[i];
   }
   printf("Enter Time Quantum:\t");
   scanf("%d",&tq);
   printf("Process ID\t\tBurst Time\t Turnaround Time\t Waiting Time");
   for(total=0,i=0;x!=0;)
   {
      if(temp[i]<=tq&&temp[i]>0)
      {
         total=total+temp[i];
         temp[i]=0;
         counter=1;
      }
      else if(temp[i]>0)
      {
         temp[i]=temp[i]-tq;
         total=total+tq;
      }
      if(temp[i]==0 && counter==1)
      {
         x--;
         printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d",i+1,bt[i],total-at[i],tota
l-at[i]-bt[i]);
         wt=wt+total-at[i]-bt[i];
         tat=tat+total-at[i];
         counter=0;
      }
      if(i==limit-1)
      {
         i=0;
      }
      else if(at[i+1]<=total)
      {
         i++;
      }
      else
      {
         i=0;
      }
   }
   awt=wt*1.0/limit;
   atat=tat*1.0/limit;
   printf("\nAverage Waiting Time:\t%f",awt);
   printf("\nAvg Turnaround Time:\t%f\n",atat);
   return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter Total Number of Processes: 3
Enter Details of Process[1]: Arrival Time: 0

Test Case - 1

```
Burst Time:      3
Enter Details of Process[2]: Arrival Time:       0
Burst Time:      2
Enter Details of Process[3]: Arrival Time:       1
Burst Time:      3
Enter Time Quantum:      5
```

Process ID	Burst Time	Turnaround Time	Waiting Time
Process[1]	3	3	0
Process[2]	2	5	3
Process[3]	3	7	4

```
Average Waiting Time:    2.333333
Avg Turnaround Time:     5.000000
```

S.No: 1	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date:

Aim:

Write a program to implement the Multi Level Queue Scheduling.

Source Code:

os5.c

```
#include<stdio.h>
#define max 20
int main()
{
   int p[max],bt[max],su[max],wt[max],tat[max],i,k,n,temp;
   float wtavg,tatavg;
   printf("Enter the number of processes:");
   scanf("%d",&n);
   for(i=0;i<n;i++){
      p[i]=i;
      printf("Enter the Burst Time of Process %d:",i);
      scanf("%d",&bt[i]);
      printf("System/User Process (0/1) ?");
      scanf("%d",&su[i]);
   }
   for(i=0;i<n;i++)
   for(k=i+1;k<n;k++)
   if(su[i]>su[k])
   {
      temp=p[i];
      p[i]=p[k];
      p[k]=temp;

      temp=bt[i];
      bt[i]=bt[k];
      bt[k]=temp;

      temp=su[i];
      su[i]=su[k];
      su[k]=temp;
   }
   wtavg=wt[0]=0;
   tatavg=tat[0]=bt[0];
   for(i=1;i<n;i++)
   {
      wt[i]=wt[i-1]+bt[i-1];
      tat[i]=tat[i-1]+bt[i];
      wtavg=wtavg+wt[i];
      tatavg=tatavg+tat[i];
   }
   printf("PROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUN
D TIME");
   for(i=0;i<n;i++)
   printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],su[i],bt[i],wt[i],tat
[i]);
   printf("\nAverage Waiting Time is --- %f",wtavg/n);
   printf("\nAverage Turnaround Time is --- %f",tatavg/n);
```

```
    return 0;
}
```

Page No:

Execution Results - All test cases have succeeded!

Test Case - 1				
User Output				
Enter the number of processes: 2				
Enter the Burst Time of Process 0: 45				
System/User Process (0/1) ? 0				
Enter the Burst Time of Process 1: 67				
System/User Process (0/1) ? 1				
PROCESS	SYSTEM/USER PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
0	0	45	0	45
1	1	67	45	112
Average Waiting Time is --- 22.500000				
Average Turnaround Time is --- 78.500000				

S.No: 3	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date:

Aim:

Write a program to implement the PRIORITY based cpu scheduling algorithm.

Source Code:

os3.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<curses.h>

void main()
{
   int n,i,j,temp,et[20],at[10],p[10],st[10],ft[10],wt[10],ta[10];
   int totwt=0,totta=0;
   float awt,ata;
   char pn[10][10],t[30];

   printf("Enter the number of process:");
   scanf("%d",&n);

   for(i=0;i<n;i++)
   {
      printf("Enter process name,arrivaltime,execution time & priority:");
      scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
   }

   for(i=0;i<n;i++)
   for(j=0;j<n;j++){
      if(at[i]<at[j]){
         temp=p[i];
         p[i]=p[j];
         p[j]=temp;

         temp=at[i];
         at[i]=at[j];
         at[j]=temp;

         temp=et[i];
         et[i]=et[j];
         et[j]=temp;

         strcpy(t,pn[i]);
         strcpy(pn[i],pn[j]);
         strcpy(pn[j],t);
      }
   }

   for(i=1;i<n;i++)
   for(j=1;j<n;j++){
      if(p[i]<p[j]){
         temp=p[i];
         p[i]=p[j];
```

```
         p[j]=temp;

         temp=at[i];
         at[i]=at[j];
         at[j]=temp;

         temp=et[i];
         et[i]=et[j];
         et[j]=temp;

         strcpy(t,pn[i]);
         strcpy(pn[i],pn[j]);
         strcpy(pn[j],t);
      }
   }

   for(i=0;i<n;i++){
      if(i==0){
         st[i]=at[i];
         wt[i]=st[i]-at[i];
         ft[i]=st[i]+et[i];
         ta[i]=ft[i]-at[i];
      }
      else{
         st[i]=ft[i-1];
         wt[i]=st[i]-at[i];
         ft[i]=st[i]+et[i];
         ta[i]=ft[i]-at[i];
      }
      totwt+=wt[i];
      totta+=ta[i];
   }
   awt=(float)totwt/n;
   ata=(float)totta/n;

   printf("Pname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime");
   for(i=0;i<n;i++){
      printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt
[i],ta[i]);
   }
   printf("\nAverage waiting time is:%f",awt);
   printf("\nAverage turnaroundtime is:%f",ata);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of process: 2
Enter process name,arrivaltime,execution time & priority: first 4 6 7
Enter process name,arrivaltime,execution time & priority: second 5 7 8
Pname    arrivaltime     executiontime    priority        waitingtime     tatime
first        4               6               7               0               6
second       5               7               8               5               12
```

Test Case - 1

```
Average waiting time is:2.500000
Average turnaroundtime is:9.000000
```

S.No: 4	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date:

Aim:

Write a program to implement the SJF Scheduling Algorithm.

Source Code:

os2.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define max 30
int main()
{
   int n,i,j,a_t[max],e_t[max],w_t[max],t_a_t[max],start[max],finish[max],temp,t
ot_t_a_t=0,tot_w_t=0;
   float awt=0,atat=0;
   char p_name[max][30],t[30];

   printf("Enter the number of process:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Enter process name, arrival time & execution time:");
      scanf("%s%d%d",p_name[i],&a_t[i],&e_t[i]);
   }
   printf("Pname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
   for(i=0;i<n;i++){
      for(j=0;j<n;j++){
         if(e_t[i]<e_t[j]){
            temp=a_t[i];
            a_t[i]=a_t[j];
            a_t[j]=temp;

            temp=e_t[i];
            e_t[i]=e_t[j];
            e_t[j]=temp;

            strcpy(t,p_name[i]);
            strcpy(p_name[i],p_name[j]);
            strcpy(p_name[j],t);
         }
      }
   }
   for(i=0;i<n;i++)
   {
      if(i==0)
         start[i]=a_t[i];
      else
         start[i]=finish[i-1];

      w_t[i]=start[i]-a_t[i];
      finish[i]=start[i]+e_t[i];
      t_a_t[i]=finish[i]-a_t[i];
      tot_w_t+=w_t[i];
      tot_t_a_t+=t_a_t[i];
```

```
   }
   awt=(float)tot_w_t/n;
   atat=(float)tot_t_a_t/n;

   for(i=0;i<n;i++){
      if(i==0)
         printf("\n%s\t   %d\t\t   %d\t\t    %d\t\t   %d",p_name[i],a_t[i],e_t
[i],w_t[i],t_a_t[i]);
      else
         printf("\n%s\t   %d\t\t   %d\t\t   %d\t\t   %d",p_name[i],a_t[i],e_t
[i],w_t[i],t_a_t[i]);
   }
   printf("\nAverage waiting time is:%f",awt);
   printf("\nAverage turnaroundtime is:%f",atat);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of process: 2
Enter process name, arrival time & execution time: first 23 24
Enter process name, arrival time & execution time: second 25 26
Pname   arrivaltime     executiontime   waitingtime     tatime
first      23              24               0              24
second     25              26              22              48
Average waiting time is:11.000000
Average turnaroundtime is:36.000000
```

S.No: 5	Exp. Name: ***Implement CPU Scheduling Algorithms***	Date:

Aim:

Write a program to implement the FCFS process scheduling algorithm.

Source Code:

os.c

```
#include<stdio.h>
#include<conio.h>
#define max 30
int main()
{
   int n,i,pn[max],at[max],bt[max],wt[max],tat[max],start[max],finish[max];
   float awt=0,atat=0;
   printf("Enter the number of processes: ");
   scanf("%d",&n);

   for(i=0;i<n;i++)
       {
          printf("Enter the Process Name, Arrival Time & Burst Time:");

          scanf("%d%d%d",&pn[i],&at[i],&bt[i]);
       }

   printf("Process Name\tArrival Time\tBurst Time\n");

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

      printf("    %d\t        %d\t        %d\n",pn[i],at[i],bt[i]);

   }

   printf("PName     Arrtime     Bursttime    Start     WT\t    TAT   Finish
\n");
   start[0]=at[0];
   finish[0]=start[0]+bt[0];
   for(i=0;i<n;i++)
   {
      if(i>0)
      {
         start[i]=finish[i-1];
      }
      finish[i]=start[i]+bt[i];
      wt[i]=start[i]-at[i];
      tat[i]=bt[i]+wt[i];

   }
   for(i=0;i<n;i++){
      if(i==0){
      printf("%d\t  %d\t\t  %d\t  %d\t   %d\t  %d\t  %d\n",pn[i],at[i],bt[i],sta
rt[i],wt[i],tat[i],finish[i]);
      }
      else{
      printf("%d\t  %d\t\t  %d\t  %d\t  %d\t  %d\t  %d\n",pn[i],at[i],bt[i],star
```

```
t[i],wt[i],tat[i],finish[i]);
      }
   }
   for(i=0;i<n;i++){
      awt+=wt[i];
      atat+=tat[i];
   }
   awt=awt/n;
   atat=atat/n;
   printf("Average Waiting time:%f\n",awt);
   printf("Average Turn Around Time:%f",atat);
   return 0;
   }
```


Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of processes:  2
Enter the Process Name, Arrival Time & Burst Time: 1 24 27
Enter the Process Name, Arrival Time & Burst Time: 1 26 27
Process Name    Arrival Time    Burst Time
    1           24              27
    1           26              27
PName     Arrtime     Bursttime    Start     WT     TAT    Finish
1         24              27      24       0      27      51
1         26              27      51      25      52      78
Average Waiting time:12.500000
Average Turn Around Time:39.500000
```

S.No: 6	Exp. Name: ***Write the code to implement Banker's Algorithm***	Date:

Aim:

Write the C program to implement Banker's Algorithm

Source Code:

bankersAlgorithm.c

```
#include<stdio.h>

#include<conio.h>

int main()

{

   int n,r,i,j,k,p,u=0,s=0,m;

   int block[10],run[10],active[10],newreq[10];

   int max[10][10],resalloc[10][10],resreq[10][10];

   int totalloc[10],totext[10],simalloc[10];

   printf("Enter the no of processes: ");

   scanf("%d",&n);

   printf("Enter the no of resource classes: ");

   scanf("%d",&r);

   printf("Enter the total existed resource in each class: ");

   for(k=1; k<=r; k++)

   scanf("%d",&totext[k]);

   printf("Enter the allocated resources: ");

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

   for(k=1; k<=r; k++)

   scanf("%d",&resalloc);

   printf("Enter the process making the new request: ");

   scanf("%d",&p);

   printf("Enter the requested resource: ");

   for(k=1; k<=r; k++)

   scanf("%d",&newreq[k]);
```

```
printf("Enter the process which are n blocked or running\n");

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

{

    if(i!=p)

     {

       printf("process %d: \n",i+1);

        scanf("%d%d",&block[i],&run[i]);

     }

}

 block[p]=0;

  run[p]=0;

   for(k=1; k<=r; k++)

    {

       j=0;

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

         {

             totalloc[k]=j+resalloc[i][k];

              j=totalloc[k];

         }

    }

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

      {

          if(block[i]==1||run[i]==1)

           active[i]=1;

            else

             active[i]=0;

      }

       for(k=1; k<=r; k++)

        {
```

```
                resalloc[p][k]+=newreq[k];

                 totalloc[k]+=newreq[k];

              }

               for(k=1; k<=r; k++)

                {

                   if(totext[k]-totalloc[k]<0)

                    {

                      u = 1;

                       break;

                    }

                }

                 if(u==0)

                  {

                      for(k=1; k<=r; k++)

                       simalloc[k]=totalloc[k];

                       for(s=1; s<=n; s++)

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

                         {

                            if(active[i]==1)

                             {

                               j=0;

                                for(k=1; k<=r; k++)

                                 {

                               if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i]
[k]))

                                {

                                   j=1;

                                    break;

                                }

                             }
```

```
                    }
                     if(j==0)
                      {
                          active[i];
                           for(k=1; k<=r; k++)
                            simalloc[k]=resalloc[i][k];
                      }
               }
                m=0;
                 for(k=1;k<=r;k++)
                  resreq[p][k]=newreq[k];
                   printf("Deadlock willn't occur\n");
}
 else
  {
    for(k=1; k<=r; k++)
     {
       resalloc[p][k]=newreq[k];
        totalloc[k]=newreq[k];
     }
      printf("Deadlock will occur\n");
  }
  return 0;
  }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the no of processes: 2

Test Case - 1
Enter the no of resource classes: 2
Enter the total existed resource in each class: 2 4 3 7
Enter the allocated resources: 5 9
Enter the process making the new request: 2 6
Enter the requested resource: 5 3
Enter the process which are n blocked or running 2 6
process 2: 2 6
Deadlock will occur

Test Case - 2
User Output
Enter the no of processes: 1
Enter the no of resource classes: 1
Enter the total existed resource in each class: 1
Enter the allocated resources: 1
Enter the process making the new request: 1
Enter the requested resource: 1
Enter the process which are n blocked or running
Deadlock willn't occur

S.No: 7	Exp. Name: ***Write the code to implement the Contiguous allocation technique: - First-Fit***	Date:

Aim:

Write a C program to implement the Contiguous allocation technique: - First-Fit

Source Code:

contiguousAllocationTechnique.c

```
#include<stdio.h>

#define max 25

void main(){

   int frag[max],b[max],f[max],i,j,nb,nf,temp;

   static int bf[max],ff[max];

   printf("Enter the number of blocks: ");

   scanf("%d",&nb);

   printf("Enter the number of files: ");

   scanf("%d",&nf);

   printf("Enter the size of the blocks\n");

   for(i=1;i<=nb;i++){

      printf("Block %d: ",i);

      scanf("%d",&b[i]);

   }

   printf("Enter the size of the files\n");

   for(i=1;i<=nf;i++){

      printf("File %d: ",i);

      scanf("%d",&f[i]);

   } for(i=1;i<=nf;i++){

      for(j=1;j<=nb;j++){

         if(bf[j]!=1){

            temp=b[j]-f[i];

            if(temp>=0){

               ff[i]=j;break;
```

```
            }
          }
        }
        frag[i]=temp;
        bf[ff[i]]=1;
    }
    printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragement\n");
    for(i=1;i<=nf;i++)
    printf("%d\t%d\t%d\t%d\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);}
```


Execution Results - All test cases have succeeded!

Test Case - 1				
User Output				
Enter the number of blocks: 3				
Enter the number of files: 2				
Enter the size of the blocks 5				
Block 1: 5				
Block 2: 1				
Block 3: 4				
Enter the size of the files 2				
File 1: 2				
File 2: 4				
File_no	File_size	Block_no	Block_size	Fragement
1	2	1	5	3
2	4	3	4	0

Test Case - 2				
User Output				
Enter the number of blocks: 4				
Enter the number of files: 6				
Enter the size of the blocks 2				
Block 1: 2				
Block 2: 6				
Block 3: 1				
Block 4: 8				
Enter the size of the files 6				
File 1: 6				
File 2: 8				
File 3: 1				
File 4: 3				
File 5: 5				

Test Case - 2

File 6: 9

File_no	File_size	Block_no	Block_size	Fragement
1	6	2	6	0
2	8	4	8	0
3	1	1	2	1
4	3	0	144	-2
5	5	0	144	-4
6	9	0	144	-8

S.No: 8	Exp. Name: ***Write a program to Implementation of Contiguous allocation technique: - Best-Fit***	Date:

ID: 1901330100189

Aim:

Write a program to Implementation of Contiguous allocation technique: - Best-Fit

Source Code:

bestFit.c

```
#include<stdio.h>

#define max 25

void main(){

   int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;

   static int bf[max],ff[max];

   printf("Memory Management Scheme for contigus memeory allocation - Best Fit
\n");

   printf("Enter the number of blocks:");

   scanf("%d",&nb);

   printf("Enter the number of files:");

   scanf("%d",&nf);

   printf("Enter the size of the blocks:-\n");

   for(i=1;i<=nb;i++){

      printf("Block %d:",i);

      scanf("%d",&b[i]);

   }

   printf("Enter the size of the files :-\n");

   for(i=1;i<=nf;i++){

      printf("File %d:",i);

      scanf("%d",&f[i]);

   }

   for(i=1;i<=nf;i++){

      for(j=1;j<=nb;j++){

         if(bf[j]!=1){

            temp=b[j]-f[i];

            if(temp>=0)

            if(lowest>temp){

               ff[i]=j;lowest=temp;

            }

         }

      }

      frag[i]=lowest;

      bf[ff[i]]=1;

      lowest=10000;

   }

   printf("File No\tFile Size \tBlock No\tBlock Size\tFragment");

   for(i=1;i<=nf && ff[i]!=0;i++)

   printf("%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

}
```

Execution Results - All test cases have succeeded!

e No:

Test Case - 1

User Output

```
Memory Management Scheme for contigus memeory allocation - Best Fit 3
Enter the number of blocks: 3
Enter the number of files: 2
Enter the size of the blocks:- 5
Block 1: 5
Block 2: 1
Block 3: 4
Enter the size of the files :- 3
File 1: 3
File 2: 4
File No File Size       Block No        Block Size      Fragment1       3               3               4               12
```

Noida Institute of Engineering and Technology

S.No: 9	Exp. Name: ***Write a program to Implementation of Contiguous allocation technique :- Worst-Fit***	Date:

Aim:

Write a program to Implementation of Contiguous allocation technique :- Worst-Fit

Source Code:

worsrFitAlgorithm.c

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
      int frag[max],b[7],f[max],i,j,nb,nf,temp,highest=0;
      static int bf[max],ff[max];
      printf("Enter the number of blocks: ");
      scanf("%d",&nb);
      printf("Enter the number of files: ");
      scanf("%d",&nf);
      printf("Enter the size of the blocks\n");
      for(i=1;i<=nb;i++)
      {
         printf("Block %d: ",i);
         scanf("%d",&b[i]);
            }
            printf("Enter the size of the files\n");
            for(i=1;i<=nf;i++)
            {
               printf("File %d: ",i);
               scanf("%d",&f[i]);
               }
               for(i=1;i<=nf;i++)
               {
                  for(j=1;j<=nb;j++)
                  {
                     if(bf[j]!=1)
                     {
                        temp=b[j]-f[i];
                        if(temp>=0)
                        if(highest<temp)
                        {
                           ff[i]=j;
                           highest=temp;
                        }
                     }
                  }




                        frag[i]=highest;
```

```
                        bf[ff[i]]=1;
                        highest=0;

                }
                   printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragement\n");
                   for(i=1;i<=nf;i++)
                   printf("%d\t%d\t%d\t%d\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);


}
```


Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the number of blocks:  4
Enter the number of files:  3
Enter the size of the blocks 5
Block 1:  5
Block 2:  4
Block 3:  3
Block 4:  5
Enter the size of the files 2
File 1:  2
File 2:  9
File 3:  4
File_no File_size       Block_no        Block_size      Fragement
1       2       1       5       3
2       9       0       0       0
3       4       4       5       1
```

Test Case - 2

User Output

```
Enter the number of blocks:  5
Enter the number of files:  7
Enter the size of the blocks 2
Block 1:  2
Block 2:  6
Block 3:  4
Block 4:  8
Block 5:  12
Enter the size of the files 36
File 1:  36
File 2:  14
File 3:  25
File 4:  4
File 5:  36
File 6:  12
File 7:  24
File_no File_size       Block_no        Block_size      Fragement
```

Test Case - 2				
1	36	0	0	0
2	14	0	0	0
3	25	0	0	0
4	4	5	12	8
5	36	0	0	0
6	12	0	0	0
7	24	0	0	0

S.No: 11	Exp. Name: ***Write a program to Implementation of contiguous memory Variable partition technique (MVT)***	Date:

Aim:

Write a program to Implementation of contiguous memory Variable partition technique (MVT)

Source Code:

VariablePartition.c

```
#include<stdio.h>

#include<conio.h>

int main(){

   int m=0,m1=0,m2=0,p,count=0,i;

   printf("enter the memory capacity:");

   scanf("%d",&m);

   printf("enter the no of processes:");

   scanf("%d",&p);

   for(i=0;i<p;i++){

      printf("enter memory req for process%d:",i+1);

      scanf("%d",&m1);

      count=count+m1;

      if(count==m)

      printf("there is no further memory remaining:\n");

      else if(m1<m){

         printf("the memory allocated for process%d is: %d ",i+1,m);

         m2=m-m1;

         printf("\nremaining memory is: %d\n",m2);

         m=m2;

      }

      else {

         printf("memory is not allocated for process%d",i+1);

      }

      printf("external fragmentation for this process is:%d\n",m2);
```

```
    }

    return 0;

}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
enter the memory capacity: 500
enter the no of processes: 2
enter memory req for process1: 250
the memory allocated for process1 is: 500 50
remaining memory is: 250 50
external fragmentation for this process is:250 50
enter memory req for process2: 50
the memory allocated for process2 is: 250
remaining memory is: 200
external fragmentation for this process is:200

Test Case - 2
User Output
enter the memory capacity: 250
enter the no of processes: 2
enter memory req for process1: 250
there is no further memory remaining: 120
external fragmentation for this process is:0 120
enter memory req for process2: 120
the memory allocated for process2 is: 250
remaining memory is: 130
external fragmentation for this process is:130

S.No: 10	Exp. Name: ***Write a program to Implementation of contiguous memory fixed partition technique(MFT)***	Date:

Aim:

Write a program to Implementation of contiguous memory fixed partition technique(MFT)

Source Code:

fixedPartitionTechnique.c

```
#include<stdio.h>
#define MAX 30
#define foi(i, lb, ub) for(int i=lb; i < ub ; i++)
int len(int n){
      int c = 0;
      while(n){
            n = n/10;
            c ++;
      }
      return c;
}
void main(){
      int l=1,ms,n,p,alm[MAX],m,frag[MAX],f=0,temp,t;
      printf("Enter the memory size:");
      scanf("%d",&ms);
      printf("Enter the no of partitions:");
      scanf("%d",&n);
      printf("Each partn size is:%dEnter the no of processes:", ms/n);
      scanf("%d",&p);
      m=ms/n;
      foi(i, 0 , p){
            printf("Enter the memory req for process%d:",i + 1);
            scanf("%d",&alm[i]);
            frag[i]=m-alm[i];
            if(frag[i] >= 0){
                  printf("Process is allocated in partition%d\n", i+1);
                  printf("Internal fragmentation for process is:%d\n", frag[i]);
            }
            else{
                  printf("Process not allocated in partition%d\n",i+1);
                  printf("External fragmentation for partition is:%d", m);
                  t=m;
                  while(t<alm[i]){
                        t=m*l;
                        l++;
                  }
                  frag[i] = t - m;

            }
            f += frag[i];
   }
   printf("Process\tmemory\tallocatedmemory\n");
   foi(i,0,p){
         printf("    ");
         printf("%d\t",i+1);
         temp=5-len(m);
         while(temp--)
```

```
            printf(" ");
        printf("%d\t",m);
        temp=5-len(alm[i]);
        while(temp--)
            printf(" ");
        printf("%d\n",alm[i]);
   }
   printf("The tot no of fragmentation is:%d", f);
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter the memory size: 500
Enter the no of partitions: 4
Each partn size is:125Enter the no of processes: 4
Enter the memory req for process1: 100
Process is allocated in partition1 200
Internal fragmentation for process is:25 200
Enter the memory req for process2: 200
Process not allocated in partition2 100
External fragmentation for partition is:125Enter the memory req for process3: 100
Process is allocated in partition3 50
Internal fragmentation for process is:25 50
Enter the memory req for process4: 50
Process is allocated in partition4
Internal fragmentation for process is:75
Process memory allocatedmemory
1 125 100
2 125 200
3 125 100
4 125 50
The tot no of fragmentation is:250