1) **Write a C program to compute average waiting time and average turn around time for First-Come First-Served (FCFS) Scheduling algorithm, the program should accept the arrival time and burst time as input.**

```
#include<stdio.h>
main()
{
float tot=0,avg,avgtr,totr=0;
int n,i,bt[100],wt[100],tr[100];
system("cls");
printf("Enter the number of process\n");
scanf("%d",&n);
printf("Enter the burst time :\n");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]);
}
wt[0]=0;
printf("Waiting time of each process\n");
printf("Waiting time of p[%d]=%d\n",0,wt[0]);
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
printf("Waiting time of p[%d]=%d\n",i,wt[i]);
}
for(i=0;i<n;i++)
{
tot=tot+wt[i];
tr[i]=wt[i]+bt[i];
printf("\n turn around time for p[%d]=%d\n",i,tr[i]);
totr=totr+tr[i];
}
```

```
avg=tot/n;
avgtr=totr/n;
printf("\n Average waiting time=%f",avg);
printf("\n Average around time%f",avgtr);
}
```

2) **Write a C program to compute average waiting time and average turn around time for Shortest-Job-First Scheduling algorithm, the program should accept the arrival time and burst time as input.**

```
#include<stdio.h>
main()
{
float tot=0,avg,avgtr,totr=0;
int i,j,n,x,bt[100],wt[100],tr[100];
system("cls");
printf("Enter the number of process:");
scanf("%d",&n);
printf("Enter burst time\n");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(bt[i]>bt[j])
{
x=bt[i];
bt[i]=bt[j];
bt[j]=x;
}
}
}
wt[0]=0;
printf("Waiting time of each process:");
printf("Waiting time of p[%d]=%d\n",0,wt[0]);
```

```
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
printf("Waiting time of p[%d]=%d\n",i,wt[i]);
}
for(i=0;i<n;i++)
{
tot=tot+wt[i];
tr[i]=wt[i]+bt[i];
printf("\n Turn around time for p[%d]=%d\n",i,tr[i]);
totr=totr+tr[i];
}
avg=tot/n;
avgtr=totr/n;
printf("\n Average waiting time=%f",avg);
printf("\n Average turn around time=%f",avgtr);
}
```

3) **Write a C program to compute average waiting time and average turn around time for Priority Scheduling algorithm, the program should accept the arrival time and burst time and priority as input.**

```
#include<stdio.h>
void main()
{
int x,n,p[10],pt[10],bt[10],wt[10],tr[10],i,j,tot=0,totr=0;
float awt,atr;
system("cls");
printf("Enter the number of process:");
scanf("%d",&n);
printf("Enter process:burst time and priority\n");
for(i=0;i<n;i++)
{
printf("\n Process number %d:",i);
scanf("%d%d",&bt[i],&pt[i]);
p[i]=i+1;
}for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(pt[i]>pt[j])
{
x=pt[i];
pt[i]=pt[j];
pt[j]=x;
x=bt[i];
bt[i]=bt[j];
bt[j]=x;
x=p[i];
p[i]=p[j];
```

```
p[j]=x;
}
}
}
wt[0]=0;
printf("Enter the waiting time of each process");
printf("\n Waiting time of p[%d]=%d\n",0,wt[0]);
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
printf("\n Waiting time p[%d]=%d",i,wt[i]);
}
for(i=0;i<n;i++)
{
tot=tot+wt[i];
tr[i]=wt[i]+bt[i];
printf("\n turn around time for p[%d]=%d",i,tr[i]);
totr=totr+tr[i];
}
awt=tot/n;
atr=totr/n;
printf("Average waiting time =%f",awt);
printf("Average turn around time =%f",atr);
}
```

4) **Write a C program to compute average waiting time and average turn around time for Round-Robin Scheduling algorithm, the program should accept the arrival time and burst time and assume suitable time quantum as input.**

```
#include<stdio.h>
void main()
{
int i,NOP,sum=0,count=0,y,quant,wt=0,tat=0,bt[10],temp[10];
float avg_wt,avg_tat;
printf("Total number of process in the system:");
scanf("%d",&NOP);
y=NOP;
for(i=0;i<NOP;i++)
{
printf("\n Enter the burst time of the process[%d]\n",i+1);
scanf("%d",&bt[i]);
temp[i]=bt[i];
}
printf("Enter the time quantum for the process:\t");
scanf("%d",&quant);
printf("\n Process no \t\t Burst time \t\t TAT \t\t Waiting time");
for(sum=0,i=0;y!=0;)
{
if(temp[i]<=quant && temp[i]>0)
{
sum=sum+temp[i];
temp[i]=0;
count=1;
}
else if(temp[i]>0)
{
temp[i]=temp[i]-quant;
```

```
sum=sum+quant;
}
if(temp[i]==0 && count==1)
{
y--;
printf("\n Process No[%d] \t\t %d\t\t\t\t%d\t\t\t%d",i+1,bt[i],sum,sum-bt[i]);
wt=wt+sum-bt[i];
tat=tat+sum;
count=0;
}
if(i==NOP-1)
{
i=0;
}
else
{
i++;
}
}
avg_wt=wt*1.0/NOP;
avg_tat=tat*1.0/NOP;
printf("\n Average turn around time:\t%f",avg_wt);
printf("\n Average waiting time:\t%f",avg_tat);
}
```

5) Write a C program for Producer-Consumer problem and hence demonstrate multi threading process.

```
#include<stdio.h>
#define max_size 3
int item[max_size];
int front,rear;
int size;
void display()
{
int k,temp;
if(size>0)
{
k=size;
temp=front;
printf("Items in buffer are\n");
while(k>0)
{
printf("%d\n",item[temp]);
temp=(temp+1)%max_size;
k--;
}
}
else
printf("Buffer is empty\n");
}
void prod()
{
if(size<max_size)
{
rear=(rear+1)%max_size;
printf("Enter the item\n");
```

```
scanf("%d",&item[rear]);
size++;
}
else
{
printf("Buffer is full\n");
display();
}
}
void cons()
{
if(size>0)
{
printf("Consumed item is =%d\n",item[front]);
front=(front+1)%max_size;
size--;
display();
}
else
printf("Buffer is empty\n");
}
void main()
{
int ch;
 size=0;
 front=0;
 rear=-1;

do
{
system("cls");
printf("\nPRODUCER CONSUMER PROBLEM\N");
```

```
printf("\n1.producer item\n2.consumer\n3.diplay\n4.exit\n");
scanf("%d",&ch);
switch(ch)
{
case 1:prod();
break;
case 2:cons();
break;
case 3:display();
break;
case 4:break;
default:printf("Invalid choice\n");
}
}
while(ch!=4);
}
```

6) Write a C program to detect whether the system is in safe state, the program should accept allocation, max and available matrices. Generate the need matrix.

```
#include<stdio.h>
int max[100][100],alloc[100][100],need[100][100],avail[100],n,r;
void input();
void cal();
void main()
{
int i,j;
printf("Bankers algorithm\n");
input();
cal();
}
void input()
{
int i,j;
printf("Enter the number of process\n");
scanf("%d",&n);
printf("Enter the number of process instance\n");
scanf("%d",&r);
printf("Enter the max matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
}
}
printf("Enter the allocation matrix\n");
for(i=0;i<n;i++)
{
```

```
for(j=0;j<r;j++)
{
scanf("%d",&alloc[i][j]);
}
}
printf("Enter the available matrix\n");
for(j=0;j<r;j++)
{
scanf("%d",&avail[i]);
}
}
void cal()
{
int finish[100],need[100][100],k,c1=0,flag=1;
int safe[100],i,j;
for(i=0;i<n;i++)
{
finish[i]=0;
}
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
need[i][j]=max[i][j]-alloc[i][j];
}
}
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
int c=0;
```

```
for(j=0;j<r;j++)
{
if((finish[i]==0)&&(need[i][j]<=avail[j]))
{
c++;
if(c==r)
{
for(k=0;k<r;k++)
{
avail[k]+=alloc[i][j];
finish[i]=1;
flag=1;
}
printf("p%d\n",i);
if(finish[i]==1)
{
i=n;
}}}}}}
for(i=0;i<n;i++)
{
if(finish[i]==1)
{
c1++;
}
else
{
printf("p%d",i);
}
}
if(c1==n)
{
printf("The system is in safe state\n");
```

```
}
else
printf("The system is in unsafe state\n");
}
```

7) Write a C program that implements FIFO page replacement algorithm.

```
#include<stdio.h>
void_print_mem(int mem[],int m)
{
int i;
for(i=0;i<m;i++)
printf("\t%d",mem[i]);
printf("\n");
}
void main()
{
int ref[25],mem[10],i,j,k,flag;
int p,m,n,pf=0;
system("cls");
printf("Enter the number of pages in the reference string");
scanf("%d",&n);
printf("Enter the frame size\n");
scanf("%d",&m);
printf("Enter the reference string\n");
for(i=0;i<n;i++)
scanf("%d",&ref[i]);
j=0;
pf=0;
for(i=0;i<m;i++)
mem[i]=-1;
for(i=0;i<n;i++)
{
p=ref[i];
flag=0;
for(k=0;k<m;k++)
if(mem[k]==p)
flag=1;
if(flag!=1)
{
mem[j]=p;
j=(j+1)%m;
pf++;
}
printf("%d.....\t",p);
void_print_mem(mem,m);
}
printf("Number of page faults=%d",pf);
}
```

8) Write a C program that implements optimal page replacement algorithm.

```
#include<stdio.h>
void_print_mem(int mem[],int m)
{
int i,optimal1,optimal2;
for(i=0;i<m;i++)
printf("\t%d",mem[i]);
printf("\n");
}
int optimal1(int p,int ref[],int c)
{
int i,n;
for(i=c+1;i<n;i++)
{
if(ref[i]==p)
return 0;
}
return -99;
}
int optimal2(int index[],int m)
{
int i,max=-99;
for(i=0;i<m;i++)
if(max<index[i])
max=index[i];
return max;
}
void main()
{
int ref[25],mem[10],index[10],i,j,k,flag;
int m,n,p,pf=0;
system("cls");
printf("Enter the number of pages in the reference string\n");
scanf("%d",&n);
printf("Enter the frame size\n");
scanf("%d",&m);
printf("Enter the reference string\n");
for(i=0;i<n;i++)
scanf("%d",&ref[i]);
j=0;
for(i=0;i<m;i++)
mem[i]=-1;
```

```
for(i=0;i<n;i++)
{
p=ref[i];
flag=0;
for(k=0;k<m;k++)
if(mem[k]==p)
{
flag=1;
break;
}
else
if(mem[k]==-1)
{
mem[k]=p;
pf++;
flag=1;
break;
}
if(flag==0)
{
pf++;
for(k=0;k<m;k++)
index[k]=optimal1(mem[k],ref,i);
for(k=0;k<m;k++)
if(index[k]==-99)
{
mem[k]=p;
flag=1;
break;
}
if(flag==0)
{
j=optimal2(index,m);
for(k=0;k<m;k++)
if(mem[k]==ref[j])
mem[k]=p;
}
}
printf("%d",p);
void_print_mem(mem,m);
}
printf("number of page fault=%d",pf);
}
```

9) Write a C program that implements LRU page replacement algorithm.

```
#include<stdio.h>
void_print_mem(int mem[],int m)
{
int i,optimal1,optimal2;
for(i=0;i<m;i++)
printf("\t%d",mem[i]);
printf("\n");
}
int optimal1(int p,int ref[],int c)
{
int i;
for(i=c-1;i>0;i--)
{
if(ref[i]==p)
return i;
}
return -99;
}
int optimal2(int index[],int m)
{
int i,min=20;
for(i=0;i<m;i++)
if(min>index[i])
min=index[i];
return min;
}
void main()
{
int ref[25],mem[10],index[10],i,j,k,flag;
int m,n,p,pf=0;
system("cls");
printf("Enter the number of pages in the references string\n");
scanf("%d",&n);
printf("Enter the frame size\n");
scanf("%d",&m);
printf("Enter the reference string\n");
for(i=0;i<n;i++)
scanf("%d",&ref[i]);
j=0;
for(i=0;i<m;i++)
```

```
mem[i]=-1;
for(i=0;i<n;i++)
{
p=ref[i];
flag=0;
for(k=0;k<m;k++)
if(mem[k]==p)
{
flag=1;
break;
}
else
if(mem[k]==-1)
{
mem[k]=p;
pf++;
flag=1;
break;
}
if(flag==0)
{
pf++;
for(k=0;k<m;k++)
index[k]=optimal1(mem[k],ref,i);
for(k=0;k<m;k++)
if(index[k]==-99)
{
mem[k]=p;
flag=1;
break;
}
if(flag==0)
{
j=optimal2(index,m);
for(k=0;k<m;k++)
if(mem[k]==ref[j])
mem[k]=p;
}
}
printf("%d",p);
void_print_mem(mem,m);
}
printf("Number of page fault=%d",pf);
}
```

10) Write a C program to implement Disk Scheduling .

```
#include<stdio.h>
void main()
{
int queue[100],t[100],head,i,n;
float seek,avg;
system("cls");
printf("FCFS disk scheduling algorithm\n");
printf("Enter the size of request queue\n");
scanf("%d",&n);
printf("Enter the elements in the queue\n");
for(i=0;i<n;i++)
scanf("%d",&queue[i]);
printf("Enter the current head position\n");
scanf("%d",&head);
for(i=0;i<n;i++)
{
t[i]=abs(head-queue[i]);
head=queue[i];
seek=seek+t[i];
}
avg=seek/n;
printf("The total seek time=%f\n",seek);
printf("Average seek time=%f",avg);
}
```