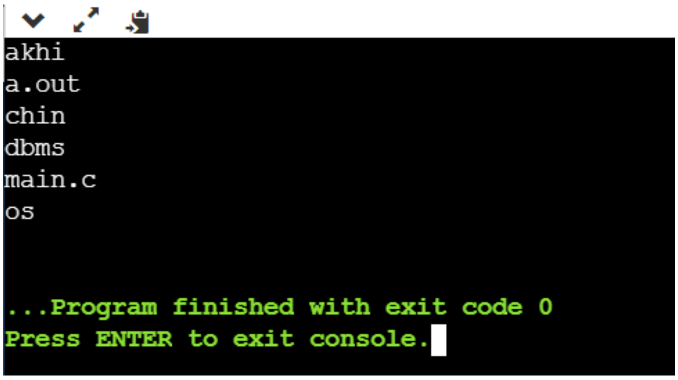
Aim:Write a c program to simulate ls|sort command

Program:

```
#include <stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main()
{
   int fd[2];
   pid_t pid;
   pipe(fd);
   pid=fork();
   if(pid>0)
   {
      close(fd[1]);
      dup2(fd[0],0);
      close(fd[0]);
      system("sort");
      exit(0);
   }
   else if(pid==0){
       close(fd[0]);
      dup2(fd[1],1);
      close(fd[1]);
      system("ls");
      exit(0);
   }
   else
   {
      printf("error in opening file");
```

```
    }


}
```

Output:



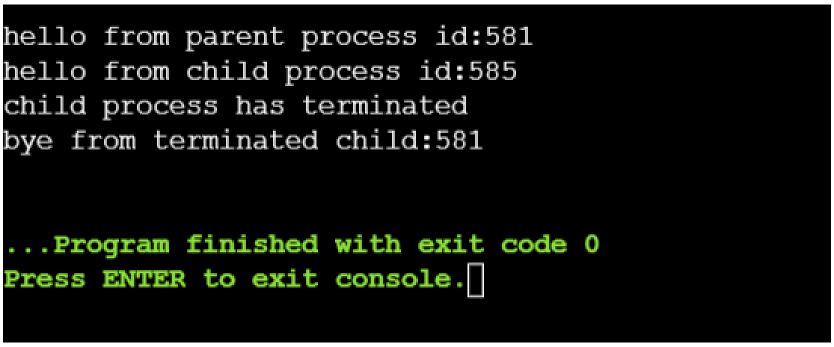
```
akhi
a.out
chin
dbms
main.c
os


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

Aim: write a c program to implement the process system calls, create a child process to it and then make it wait and abort.

Program:

```
#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h>

int main(){

        if(fork()==0){

                printf("hello from child process id:%d\n",getpid());

                abort();

        }

        else

        {

        printf("hello from parent process id:%d\n",getpid());

        wait(NULL);

        printf("child process has terminated\n");

        }

        printf("bye from terminated child:%d\n",getpid());

        return 0;

}
```

Output:



```
hello from parent process id:581
hello from child process id:585
child process has terminated
bye from terminated child:581


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

```
}
```

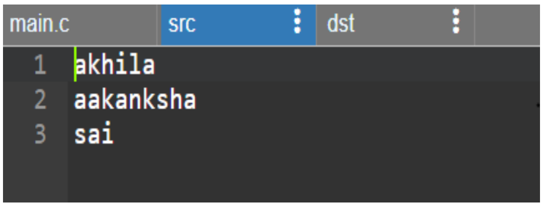
Aim: write a c program to simulate the contents of one file to another file using system calls

Program:

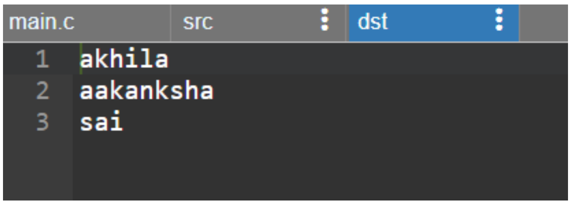
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<string.h>
#include<unistd.h>
void main()
{
   int n,fd,fd1;
   char filename[100],b[1024];
   //open one file for reading
   printf("enter the filename to open for reading:\n");
   scanf("%s",filename);
   fd=open(filename,0);//opening sourcefile
   if(fd==-1)
   {
      printf("error in opening file");
                  exit(0);
   }
   //open other file for writing
   printf("enter the filename to open for writing\n");
   scanf("%s",filename);
   fd1=open(filename,1); //opening destination file
    if(fd1==-1)
   {
      printf("error in opening file");
                  exit(0);
   }
```

```
    n=read(fd,b,sizeof(b)); //reading the contents of source file


    write(fd1,b,n); //writing the contents to destination file

    printf("file copied successfully");

    close(fd);

    close(fd1);


}
```

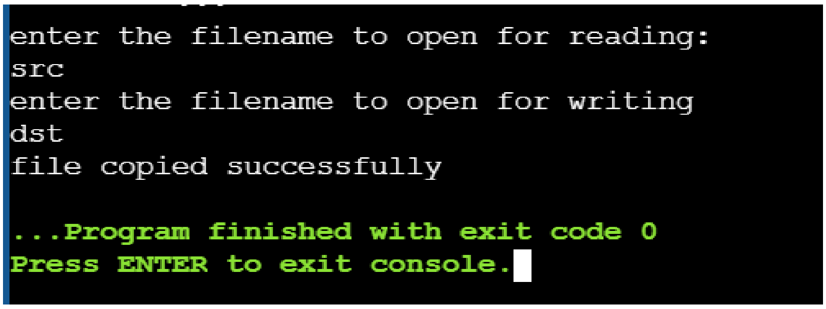
Output:



```
akhila
aakanksha
sai
```



```
akhila
aakanksha
sai
```



```
enter the filename to open for reading:
src
enter the filename to open for writing
dst
file copied successfully

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

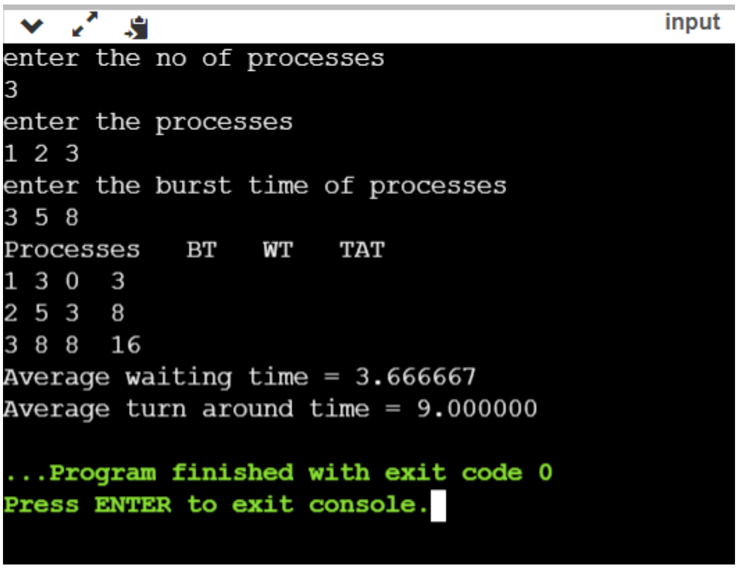
Aim: write a c program to simulate FCFS scheduling algorithm without arrival time.

Program:

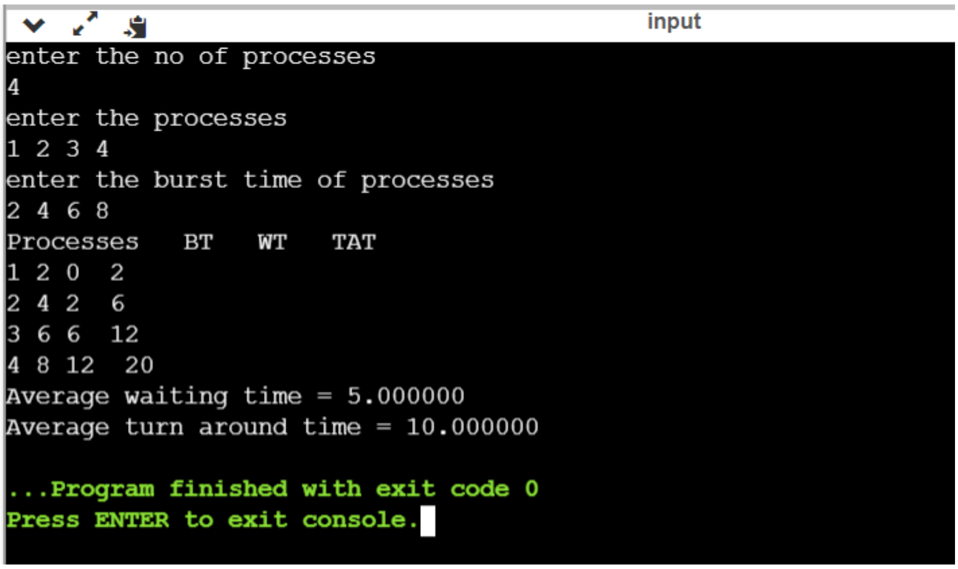
```
 #include <stdio.h>
void WT(int processes[],int n,int bt[],int wt[])
{
   wt[0]=0;
   for(int i=1;i<n;i++)
      wt[i]=bt[i-1]+wt[i-1];
}
void TAT(int processes[],int n,int bt[],int wt[],int tat[])
{
   for(int i=0;i<n;i++)
      tat[i]=bt[i]+wt[i];
}
void AverageTime(int processes[],int n,int bt[])
{
   int wt[n],tat[n],total_wt = 0,total_tat = 0;

   WT(processes,n,bt,wt);
   TAT(processes,n,bt,wt,tat);
   printf("Processes   BT   WT   TAT\n");
   for (int  i=0; i<n; i++)
   {
      total_wt = total_wt + wt[i];
      total_tat = total_tat + tat[i];
      printf("%d ",(i+1));
      printf("%d ", bt[i]);
      printf("%d",wt[i]);
      printf("  %d\n",tat[i]);
```

```
    }
    float s=(float)total_wt/n;
    float t=(float)total_tat/n;
    printf("Average waiting time = %f",s);
    printf("\n");
    printf("Average turn around time = %f",t);
}
int main()
{
    int p;
    printf("enter the no of processes\n");
    scanf("%d",&p);
    int processes[p];
    printf("enter the processes\n");
    for(int i=0;i<p;i++)
        scanf("%d",&processes[i]);
    int n=sizeof processes/sizeof processes[0];
    int bt[n];
    printf("enter the burst time of processes\n");
     for(int i=0;i<n;i++)
        scanf("%d",&bt[i]);
    AverageTime(processes,n,bt);
    return 0;
}
```

Output:



```
enter the no of processes
3
enter the processes
1 2 3
enter the burst time of processes
3 5 8
Processes    BT    WT    TAT
1 3 0   3
2 5 3   8
3 8 8   16
Average waiting time = 3.666667
Average turn around time = 9.000000

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



```
enter the no of processes
4
enter the processes
1 2 3 4
enter the burst time of processes
2 4 6 8
Processes    BT    WT    TAT
1 2 0   2
2 4 2   6
3 6 6   12
4 8 12   20
Average waiting time = 5.000000
Average turn around time = 10.000000

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

Aim:write a c program to simulate FCFS scheduling algorithm with arrival time

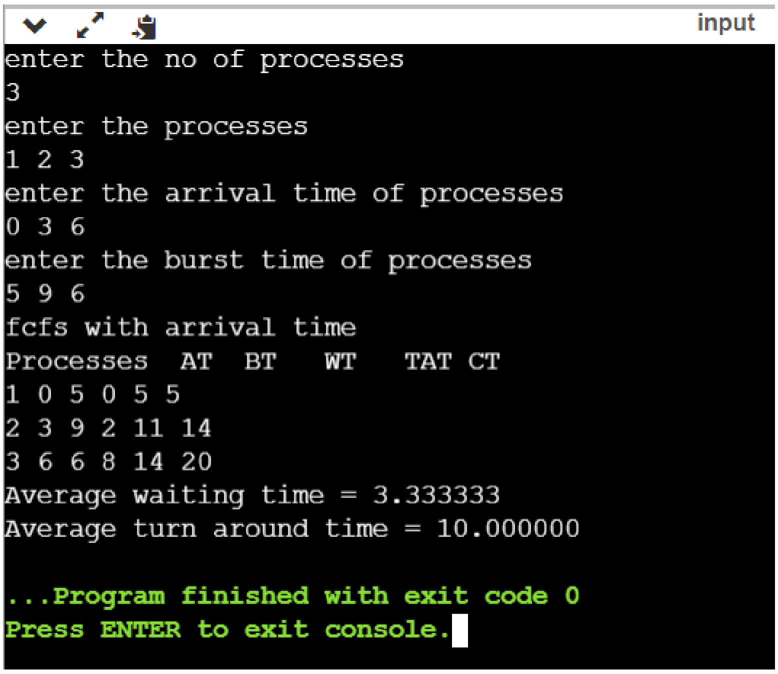
Program:

```
#include <stdio.h>
void WT(int processes[],int n,int bt[],int wt[],int at[])
{
   int servicetime[n],i;
   servicetime[0]=at[0];
   wt[0]=0;
   for(i=1;i<n;i++)
   {
     servicetime[i]=servicetime[i-1]+bt[i-1];
     wt[i]=servicetime[i]-at[i];
      if(wt[i]<0)
          wt[i]=0;
   }
}
void TAT(int processes[],int n,int bt[],int wt[],int tat[])
{
         int i;
   for(i=0;i<n;i++)
     tat[i]=bt[i]+wt[i];
}
void AverageTime(int processes[],int n,int bt[],int at[])
{
   int wt[n],tat[n],i;
   WT(processes,n,bt,wt,at);
   TAT(processes,n,bt,wt,tat);
   printf("Processes  AT  BT   WT   TAT CT\n");
   int total_wt = 0,total_tat = 0;
```

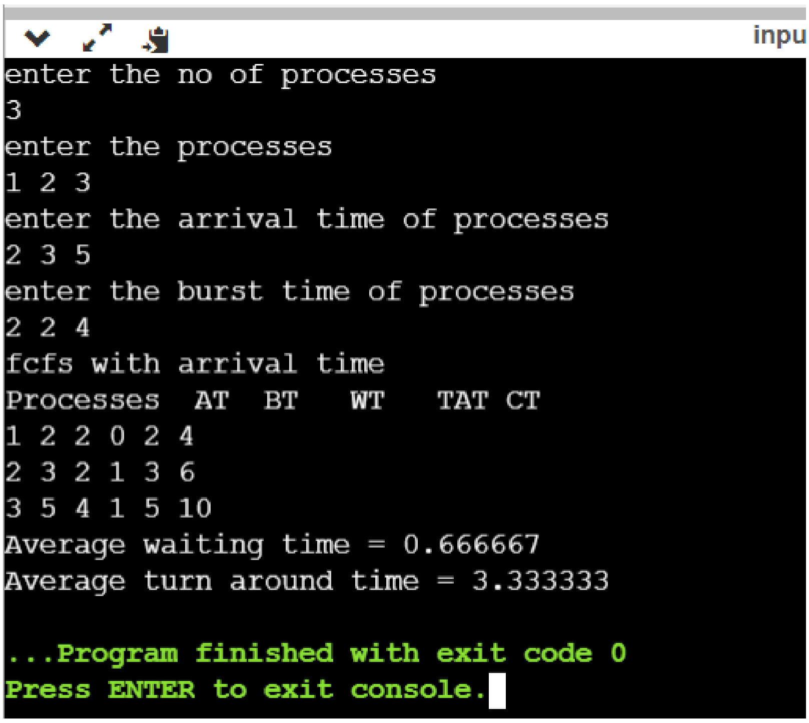
```
    for (i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        int ct=tat[i]+at[i];
        printf("%d ",(i+1));
        printf("%d ",at[i]);
        printf("%d ",bt[i]);
        printf("%d ",wt[i]);
        printf("%d ",tat[i]);
        printf("%d\n",ct);
    }
    float s=(float)total_wt/n;
    float t=(float)total_tat/n;
    printf("Average waiting time = %f",s);
    printf("\n");
    printf("Average turn around time = %f",t);
}
int main()
{
    int p,i;
    printf("enter the no of processes\n");
    scanf("%d",&p);
    int processes[p];
    printf("enter the processes\n");
    for(i=0;i<p;i++)
        scanf("%d",&processes[i]);
    int n=sizeof processes/sizeof processes[0];
    int at[p];
```

```
   printf("enter the arrival time of processes\n");
   for(i=0;i<p;i++)
      scanf("%d",&at[i]);
   int bt[p];
   printf("enter the burst time of processes\n");
   for(i=0;i<p;i++)
      scanf("%d",&bt[i]);
   printf("fcfs with arrival time \n");
   AverageTime(processes,n,bt,at);
   return 0;
}
```

Output:



```
enter the no of processes
3
enter the processes
1 2 3
enter the arrival time of processes
0 3 6
enter the burst time of processes
5 9 6
fcfs with arrival time
Processes  AT  BT   WT   TAT CT
1 0 5 0 5 5
2 3 9 2 11 14
3 6 6 8 14 20
Average waiting time = 3.333333
Average turn around time = 10.000000

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



```
enter the no of processes
3
enter the processes
1 2 3
enter the arrival time of processes
2 3 5
enter the burst time of processes
2 2 4
fcfs with arrival time
Processes  AT  BT   WT   TAT CT
1 2 2 0 2 4
2 3 2 1 3 6
3 5 4 1 5 10
Average waiting time = 0.666667
Average turn around time = 3.333333

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

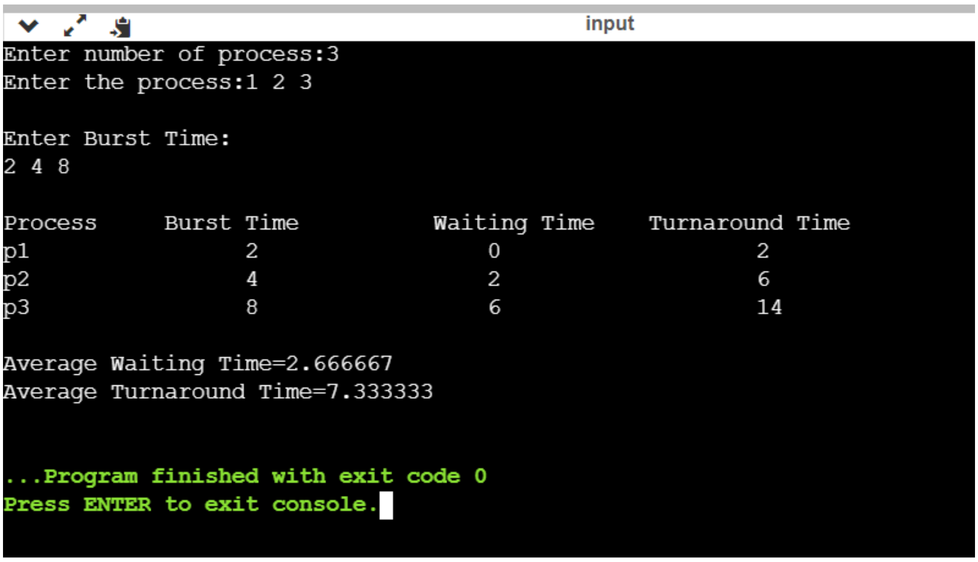
Aim:write a c program to simulate SJF scheduling algorithm without arrival time

Program:

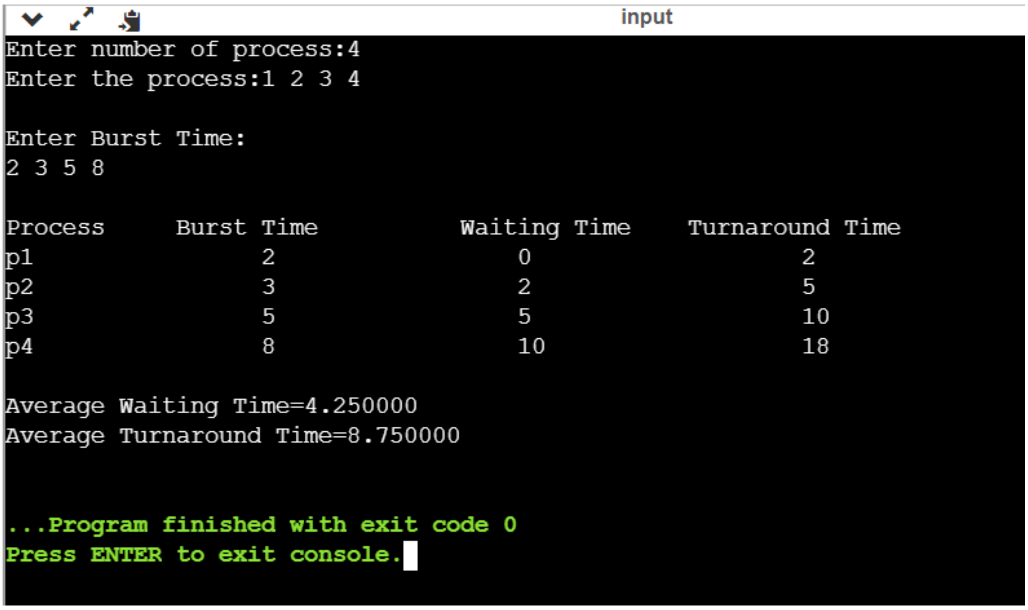
```
 #include<stdio.h>
void main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process:");
   scanf("%d",&n);
   int processes[n];
   printf("Enter the process:");
   for(i=0;i<n;i++)
     scanf("%d",&processes[i]);
   printf("\nEnter Burst Time:\n");
   for(i=0;i<n;i++)
      scanf("%d",&bt[i]);
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(bt[j]<bt[pos])
             pos=j;
      }

      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;
```

```
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=(float)total/n;
    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",(i+1),bt[i],wt[i],tat[i]);
    }

    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output:



```
Enter number of process:3
Enter the process:1 2 3

Enter Burst Time:
2 4 8

Process      Burst Time          Waiting Time     Turnaround Time
p1                 2                  0                  2
p2                 4                  2                  6
p3                 8                  6                  14

Average Waiting Time=2.666667
Average Turnaround Time=7.333333


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



```
Enter number of process:4
Enter the process:1 2 3 4

Enter Burst Time:
2 3 5 8

Process      Burst Time          Waiting Time     Turnaround Time
p1                 2                  0                  2
p2                 3                  2                  5
p3                 5                  5                  10
p4                 8                  10                 18

Average Waiting Time=4.250000
Average Turnaround Time=8.750000


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

Aim:write a c program to simulate SJF scheduling algorithm with arrival time

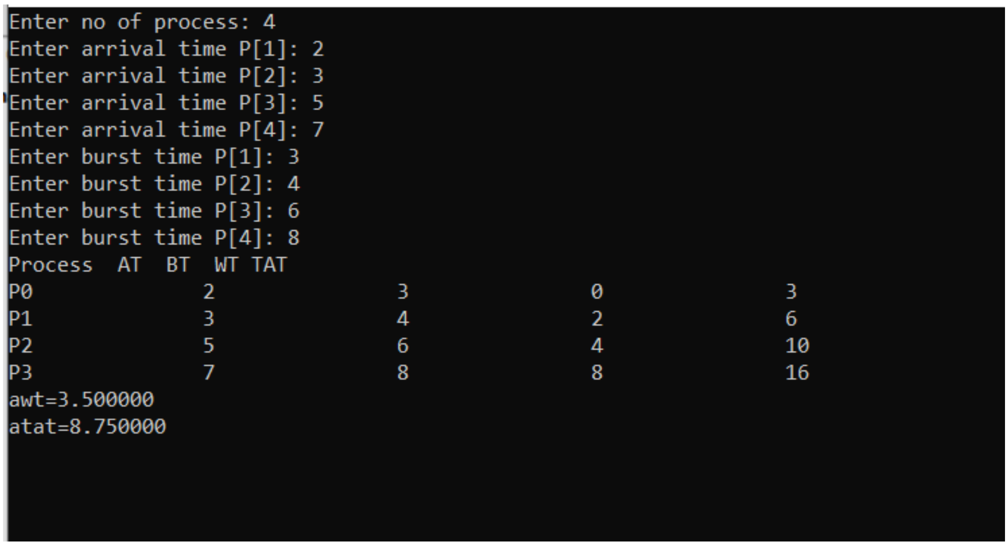
Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,temp,tt=0,min,d,i,j;
    float atat=0,awt=0,stat=0,swt=0;
    printf("Enter no of process: ");
    scanf("%d",&n);
    int a[10],b[10],e[10],tat[10],wt[10];
    for(i=0;i<n;i++)
    {
   printf("Enter arrival time P[%d]: ",i+1);
   scanf("%d",&a[i]);
    }
    for(i=0;i<n;i++)
    {
   printf("Enter burst time P[%d]: ",i+1);
   scanf("%d",&b[i]);
    }
    for(i=0;i<n;i++)
    {
      for(j=i+1;j<n;j++)
        {
                  if(b[i]>b[j])
                  {
                  temp=a[i];
                  a[i]=a[j];
```

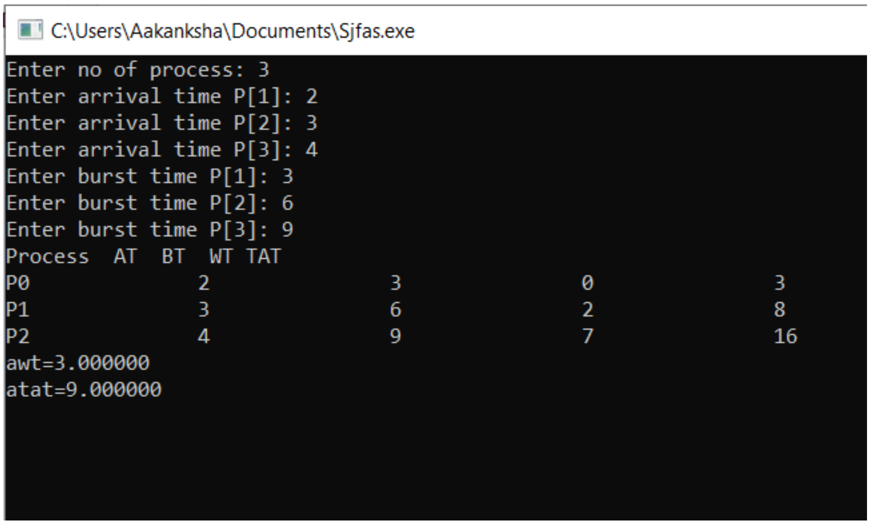
```
                a[j]=temp;

                temp=b[i];

                b[i]=b[j];

                b[j]=temp;

        }


      }

     }

     min=a[0];

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

     {

         if(min>a[i])

         {

              min=a[i];

              d=i;

         }

     }

     tt=min;

     e[d]=tt+b[d];

     tt=e[d];

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

     {

         if(a[i]!=min)

         {

              e[i]=b[i]+tt;

              tt=e[i];

         }

     }

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

```
   {
      tat[i]=e[i]-a[i];
      stat=stat+tat[i];
      wt[i]=tat[i]-b[i];
      swt=swt+wt[i];
   }
   atat=stat/n;
   awt=swt/n;
   printf("Process  AT  BT  WT TAT \n");
  for(i=0;i<n;i++)
  {
  printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,a[i],b[i],wt[i],tat[i]);
  }
  printf("awt=%f\natat=%f",awt,atat);
  getch();

}
```

Output:



```
Enter no of process: 4
Enter arrival time P[1]: 2
Enter arrival time P[2]: 3
Enter arrival time P[3]: 5
Enter arrival time P[4]: 7
Enter burst time P[1]: 3
Enter burst time P[2]: 4
Enter burst time P[3]: 6
Enter burst time P[4]: 8
Process  AT  BT  WT TAT
P0              2               3               0               3
P1              3               4               2               6
P2              5               6               4               10
P3              7               8               8               16
awt=3.500000
atat=8.750000
```



C:\Users\Aakanksha\Documents\Sjfas.exe

```
Enter no of process: 3
Enter arrival time P[1]: 2
Enter arrival time P[2]: 3
Enter arrival time P[3]: 4
Enter burst time P[1]: 3
Enter burst time P[2]: 6
Enter burst time P[3]: 9
Process  AT  BT  WT TAT
P0              2               3               0               3
P1              3               6               2               8
P2              4               9               7               16
awt=3.000000
atat=9.000000
```

```
}
```

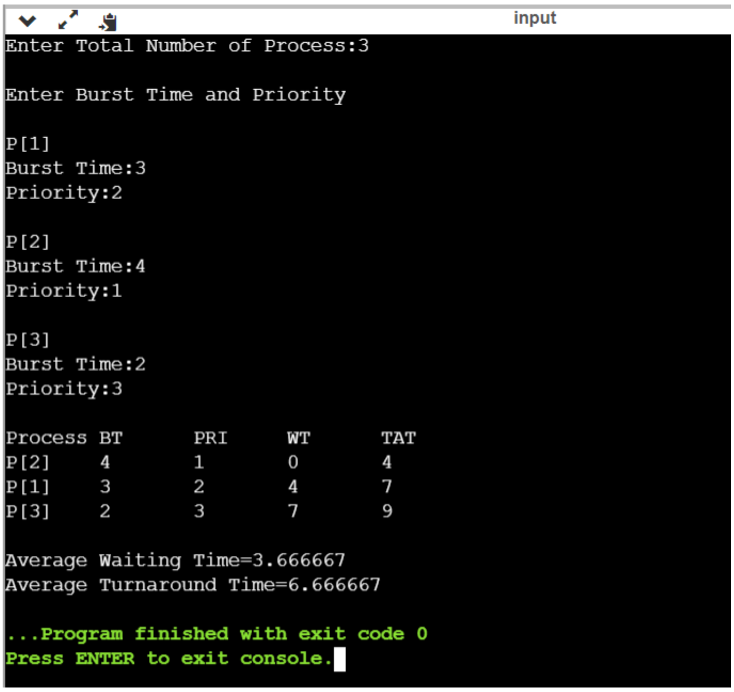
Aim:write a c program to simulate PRIORITY scheduling algorithm without arrival time

Program:

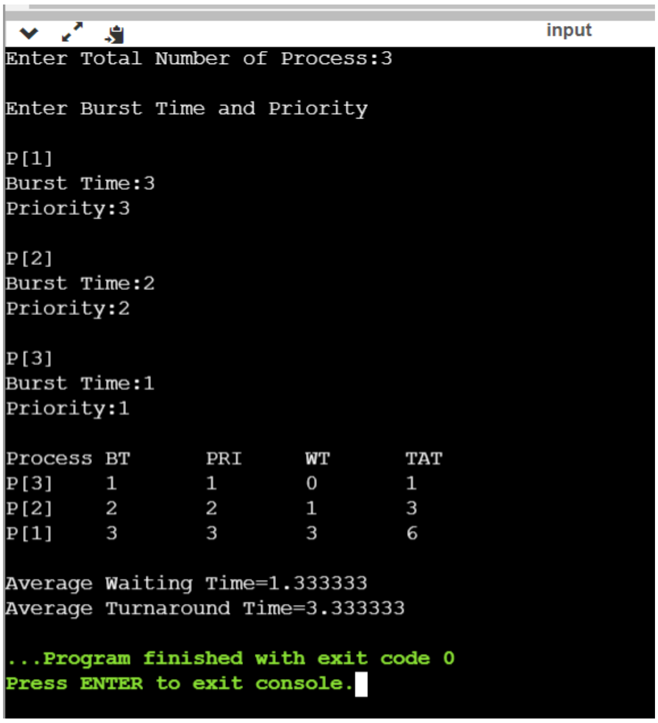
```
#include<stdio.h>
int main()
{
  int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp;
      float avg_wt,avg_tat;
  printf("Enter Total Number of Process:");
  scanf("%d",&n);
  printf("\nEnter Burst Time and Priority\n");
  for(i=0;i<n;i++)
  {
    printf("\nP[%d]\n",i+1);
    printf("Burst Time:");
    scanf("%d",&bt[i]);
    printf("Priority:");
    scanf("%d",&pr[i]);
    p[i]=i+1;
  }
  for(i=0;i<n;i++)
  {
    pos=i;
    for(j=i+1;j<n;j++)
    {
      if(pr[j]<pr[pos])
        pos=j;
    }

    temp=pr[i];
    pr[i]=pr[pos];
    pr[pos]=temp;
    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;
    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}
wt[0]=0;
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
    total+=wt[i];
}
avg_wt=(float)total/n;
total=0;
printf("\nProcess\tBT\tPRI\tWT\tTAT");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    total+=tat[i];
    printf("\nP[%d]\t%d\t%d\t%d\t%d",p[i],bt[i],pr[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f",avg_tat);
```

```
        return 0;
}
```

Output:



```
Enter Total Number of Process:3

Enter Burst Time and Priority

P[1]
Burst Time:3
Priority:2

P[2]
Burst Time:4
Priority:1

P[3]
Burst Time:2
Priority:3

Process BT      PRI     WT      TAT
P[2]    4       1       0       4
P[1]    3       2       4       7
P[3]    2       3       7       9

Average Waiting Time=3.666667
Average Turnaround Time=6.666667

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



```
Enter Total Number of Process:3

Enter Burst Time and Priority

P[1]
Burst Time:3
Priority:3

P[2]
Burst Time:2
Priority:2

P[3]
Burst Time:1
Priority:1

Process BT      PRI     WT      TAT
P[3]    1       1       0       1
P[2]    2       2       1       3
P[1]    3       3       3       6

Average Waiting Time=1.333333
Average Turnaround Time=3.333333

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

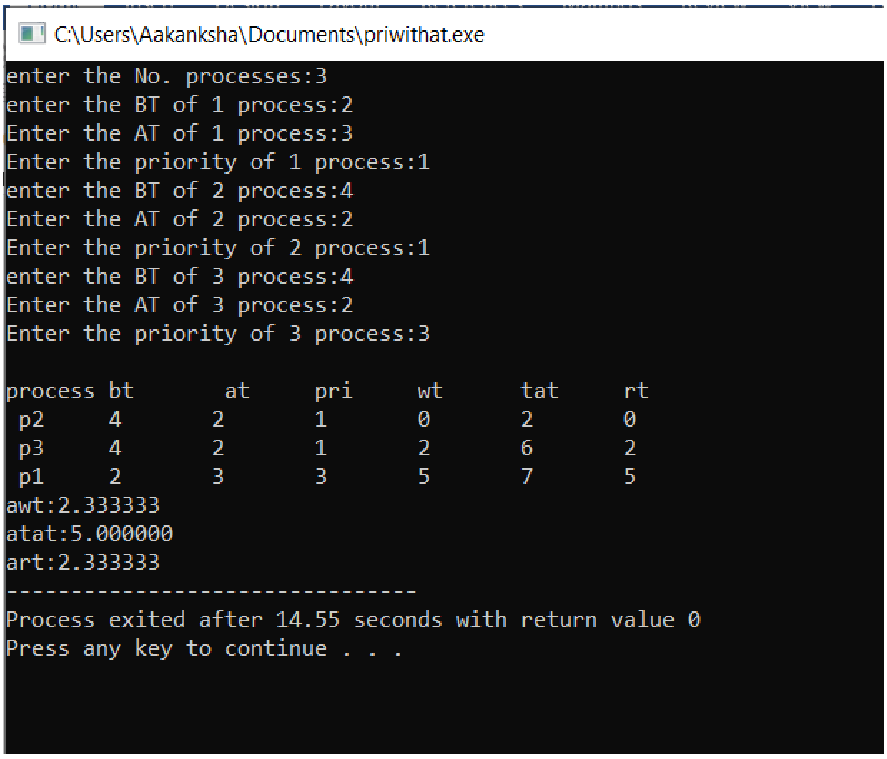
Aim:write a c program to simulate PRIORITY scheduling algorithm with arrival time

Program:

```
#include<stdio.h>
int main()
{
	int i,n,p[10]={1,2,3,4,5,6,7,8,9,10},min,k=1,burst=0,pri[10];
	int bt[10],temp,temp1,j,at[10],wt[10],rt[10],tt[10],ta=0,sum=0;
	float wavg,tavg,tsum,wsum;
	printf("enter the No. processes:");
	scanf("%d",&n);
	for(i=0;i<n;i++) {
	printf("enter the BT of %d process:",i+1);
	scanf("%d",&bt[i]);
	printf("Enter the AT of %d process:",i+1);
	scanf("%d",&at[i]);
	printf("Enter the priority of %d process:",i+1);
	scanf("%d",&pri[i]);
	}
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
		{
			if(at[i]<at[j])/*sorting acc to arrival time*/
			{
			temp=p[j];
			p[j]=p[i];
			p[i]=temp;
			temp=at[j];
			at[j]=at[i];
```

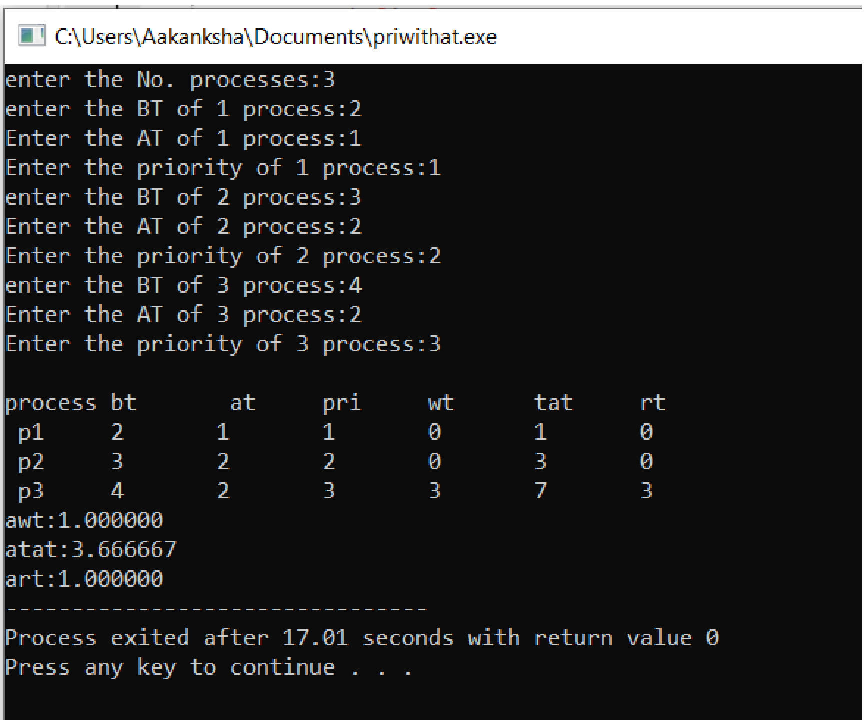
```
                    at[i]=temp;
                    temp1=bt[j];
                    bt[j]=bt[i];
                    bt[i]=temp1;
                    }
          }
}
for(j=0;j<n;j++)
{
          burst=burst+bt[j];
          min=bt[k];
          for(i=k;i<n;i++)/*main logic*/
          {
                    min=pri[k];
                    if (burst>=at[i])
                    {
                              if(pri[i]<min)
                              {
                              temp=p[k];
                              p[k]=p[i];
                              p[i]=temp;
                              temp=at[k];
                              at[k]=at[i];
                              at[i]=temp;
                              temp1=bt[k];
                              bt[k]=bt[i];
                              bt[i]=temp1;
                              temp=pri[k];
                              pri[k]=pri[i];
                    pri[i]=temp;
                    }
               }
          }
          k++;
     }
     wt[0]=0;
     for(i=1;i<n;i++)
     {
          sum=sum+bt[i-1];
          wt[i]=sum-at[i];
     }
     for(i=0;i<n;i++)
     {
          wsum=wsum+wt[i];
     }
     wavg=wsum/n;
     for(i=0;i<n;i++)
     {
          ta=ta+bt[i];
          tt[i]=ta-at[i];
     }
     for(i=0;i<n;i++)
     {
          tsum=tsum+tt[i];
     }
     tavg=tsum/n;
     for(i=0;i<n;i++)
     {
            rt[i]=wt[i];
        }
        printf("\nprocess\tbt\t at\tpri\twt\ttat\trt" );
        for(i=0;i<n;i++)
        {
        printf("\n p%d\t%d\t%d\t%d\t%d\t%d\t%d",p[i],bt[i],at[i],pri[i],wt[i],tt[i],rt[i]);
        }
        printf("\nawt:%f",wavg);
        printf("\natat:%f",tavg);
        printf("\nart:%f",wavg);
}
```

Output:



C:\Users\Aakanksha\Documents\priwithat.exe

```
enter the No. processes:3
enter the BT of 1 process:2
Enter the AT of 1 process:3
Enter the priority of 1 process:1
enter the BT of 2 process:4
Enter the AT of 2 process:2
Enter the priority of 2 process:1
enter the BT of 3 process:4
Enter the AT of 3 process:2
Enter the priority of 3 process:3

process bt       at      pri     wt      tat     rt
 p2     4       2       1       0       2       0
 p3     4       2       1       2       6       2
 p1     2       3       3       5       7       5
awt:2.333333
atat:5.000000
art:2.333333
--------------------------------
Process exited after 14.55 seconds with return value 0
Press any key to continue . . .
```



C:\Users\Aakanksha\Documents\priwithat.exe

```
enter the No. processes:3
enter the BT of 1 process:2
Enter the AT of 1 process:1
Enter the priority of 1 process:1
enter the BT of 2 process:3
Enter the AT of 2 process:2
Enter the priority of 2 process:2
enter the BT of 3 process:4
Enter the AT of 3 process:2
Enter the priority of 3 process:3

process bt       at      pri     wt      tat     rt
 p1     2       1       1       0       1       0
 p2     3       2       2       0       3       0
 p3     4       2       3       3       7       3
awt:1.000000
atat:3.666667
art:1.000000
--------------------------------
Process exited after 17.01 seconds with return value 0
Press any key to continue . . .
```

Aim:write a c program to simulate roundrobin scheduling algorithm

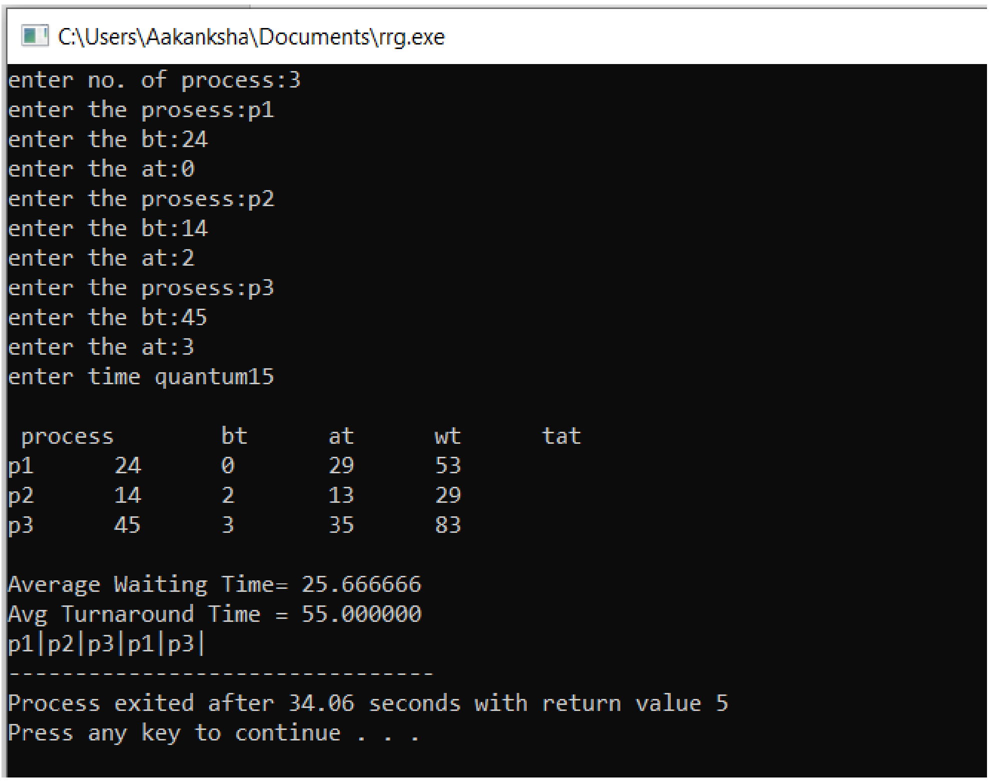
Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
	char proc[100][3],gant[100][3],temp[20];
	int 
n,gn=0,tq,time=0,tbtime[100],btime[100],arrtime[100],wtime[100]={0},tat[100]={0},i,k=0,c=0;
	float wsum=0,tsum=0;
	printf("enter no. of process:");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("enter the prosess:");
		scanf("%s",&proc[i]);
		printf("enter the bt:");
		scanf("%d",&btime[i]);
		printf("enter the at:");
		scanf("%d",&arrtime[i]);
		tbtime[i]=btime[i];
	}
	printf("enter time quantum:");
	scanf("%d",&tq);
	while(c<n)
	{
		if(btime[k]>0)
		{
			if(c<n-1)
```

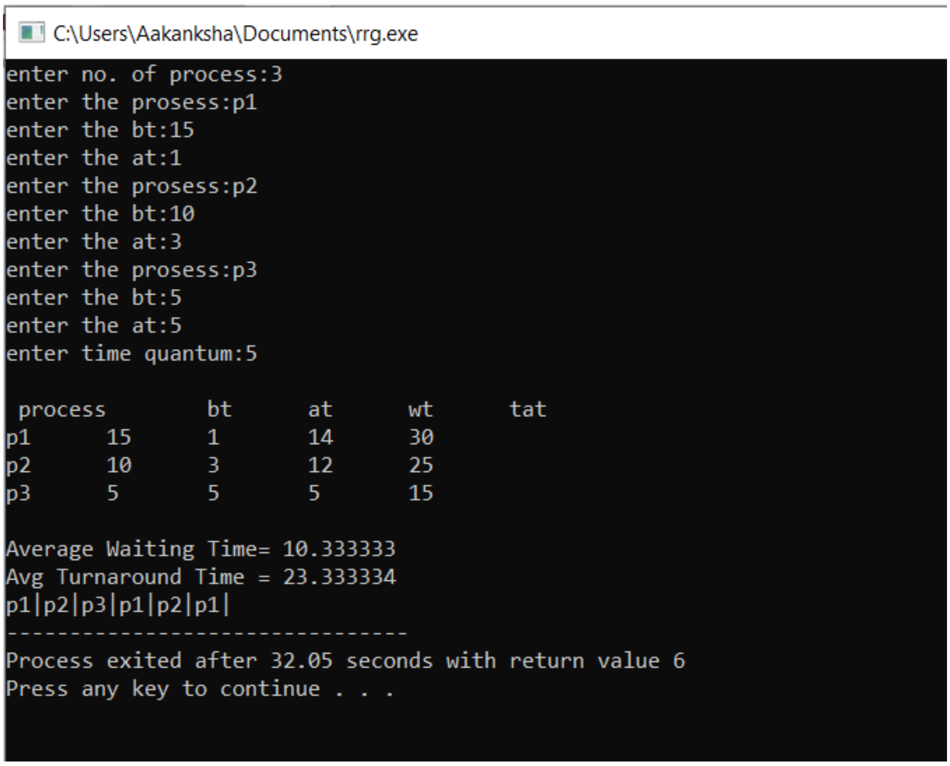
```
                {
                        strcpy(gant[gn],proc[k]);
                        gn++;
                }
                else
                {
                        strcpy(temp,proc[k]);
                }
                if(btime[k]-tq>0)
                {
                        time+=tq;
                }
                else
                {
                        time+=btime[k];
                        tat[k]=time;
                        c++;
                }
                btime[k]-=tq;
        }
        k++;
        if(k==n)
        {
                k=0;
        }
}
printf("\n process\tbt\tat\twt\ttat\n");
for(i=0;i<n;i++)
{
```

```
                wtime[i]=tat[i]-tbtime[i]-arrtime[i];
                wsum+=wtime[i];
                tsum+=tat[i];
          printf("%s\t%d\t%d\t%d\t%d\n",proc[i],tbtime[i],arrtime[i],wtime[i],tat[i]);
        }
         printf("\nAverage Waiting Time= %f\n",wsum/n);
    printf("Avg Turnaround Time = %f\n",tsum/n);
         strcpy(gant[gn++],temp);
         for(i=0;i<gn;i++)
         {
                printf("%s|",gant[i]);
         }
}
```

Output:



```
enter no. of process:3
enter the prosess:p1
enter the bt:24
enter the at:0
enter the prosess:p2
enter the bt:14
enter the at:2
enter the prosess:p3
enter the bt:45
enter the at:3
enter time quantum15

 process        bt      at      wt      tat
p1      24      0       29      53
p2      14      2       13      29
p3      45      3       35      83

Average Waiting Time= 25.666666
Avg Turnaround Time = 55.000000
p1|p2|p3|p1|p3|
--------------------------------
Process exited after 34.06 seconds with return value 5
Press any key to continue . . .
```



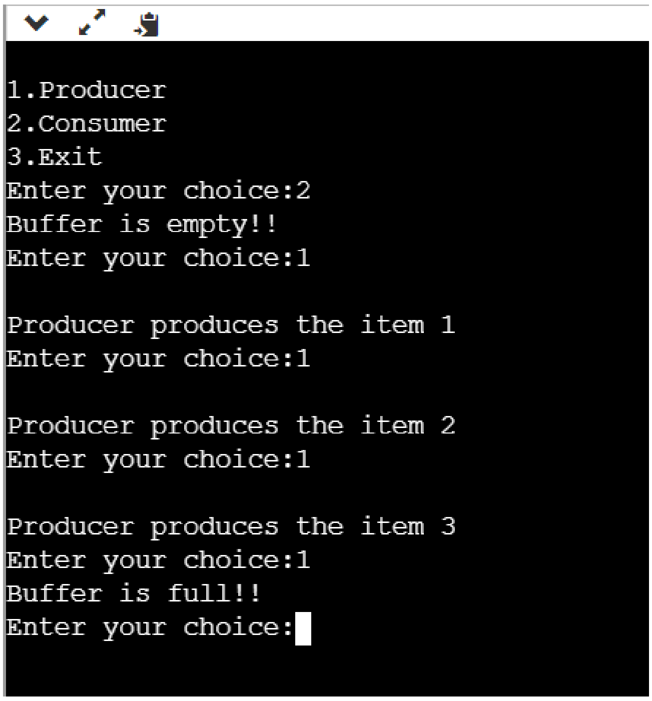
```
enter no. of process:3
enter the prosess:p1
enter the bt:15
enter the at:1
enter the prosess:p2
enter the bt:10
enter the at:3
enter the prosess:p3
enter the bt:5
enter the at:5
enter time quantum:5

 process        bt      at      wt      tat
p1      15      1       14      30
p2      10      3       12      25
p3      5       5       5       15

Average Waiting Time= 10.333333
Avg Turnaround Time = 23.333334
p1|p2|p3|p1|p2|p1|
--------------------------------
Process exited after 32.05 seconds with return value 6
Press any key to continue . . .
```

Aim:Write a c program to solve producer consumer problem using semaphore

Program:

```
#include<stdio.h>
#include<conio.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);//producer produes an item
   full=signal(full);//produce and item & full is increased by 1
   //critical section
   empty=wait(empty);//empty is reduced by 1 beoz 1 slot is filled
   x++;
   printf("\nProducer produces the item %d",x);
   mutex=signal(mutex);//now consumer can access the buffer
}

void consumer()
{
   mutex=wait(mutex);//reduced by 1 beoz producer cant access buffer
   full=wait(full); //reduced by 1 becoz consumer consumes an item
   empty=signal(empty);//consumer consumes the item ,increased by 1
   printf("\nConsumer consumes item %d",x);
   x--;
   mutex=signal(mutex);//producer can access the buffer
}
```

Output:



```
1.Producer
2.Consumer
3.Exit
Enter your choice:2
Buffer is empty!!
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:
```

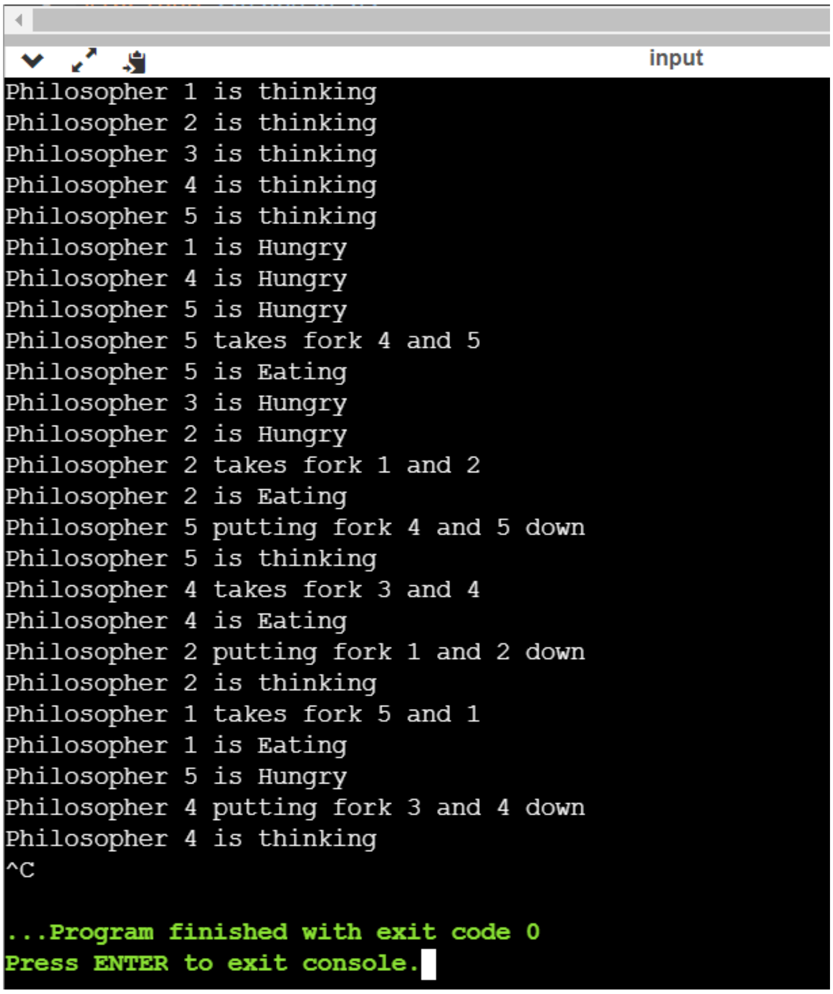
Aim: write a c program to solve dining philosopher problem using monitor

Program:

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <ctype.h>
#include <semaphore.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N
int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };
sem_t mutex;
sem_t S[N];
void test(int phnum)
{
   if (state[phnum] == HUNGRY
      && state[LEFT] != EATING
      && state[RIGHT] != EATING) {
      // state that eating
      state[phnum] = EATING;
      sleep(2);
      printf("Philosopher %d takes fork %d and %d\n",
               phnum + 1, LEFT + 1, phnum + 1);

      printf("Philosopher %d is Eating\n", phnum + 1);
```

```
        // sem_post(&S[phnum]) has no effect
        // during takefork
        // used to wake up hungry philosophers
        // during putfork
        sem_post(&S[phnum]);
    }
}
// take up chopsticks
void take_fork(int phnum)
{
    sem_wait(&mutex);
    // state that hungry
    state[phnum] = HUNGRY;
    printf("Philosopher %d is Hungry\n", phnum + 1);
    // eat if neighbours are not eating
    test(phnum);
    sem_post(&mutex);
    // if unable to eat wait to be signalled
    sem_wait(&S[phnum]);
    sleep(1);
}
// put down chopsticks
void put_fork(int phnum)
{
    sem_wait(&mutex);
    // state that thinking
    state[phnum] = THINKING;
    printf("Philosopher %d putting fork %d and %d down\n",
           phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}
void* philospher(void* num)
{
    while (1) {
        int* i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}
int main()
{
    int i;
    pthread_t thread_id[N];

    // initialize the semaphores
    sem_init(&mutex, 0, 1);
    for (i = 0; i < N; i++)
        sem_init(&S[i], 0, 0);
    for (i = 0; i < N; i++)
    {
        // create philosopher processes
        pthread_create(&thread_id[i], NULL,philospher, &phil[i]);
        printf("Philosopher %d is thinking\n", i + 1);
    }
    for (i = 0; i < N; i++)
        pthread_join(thread_id[i], NULL);
}
```

Output:



```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 3 is Hungry
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 5 is Hungry
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
^C

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

Aim: write a c program to simulate bankers algorithm for deadlock avoidance

Program:

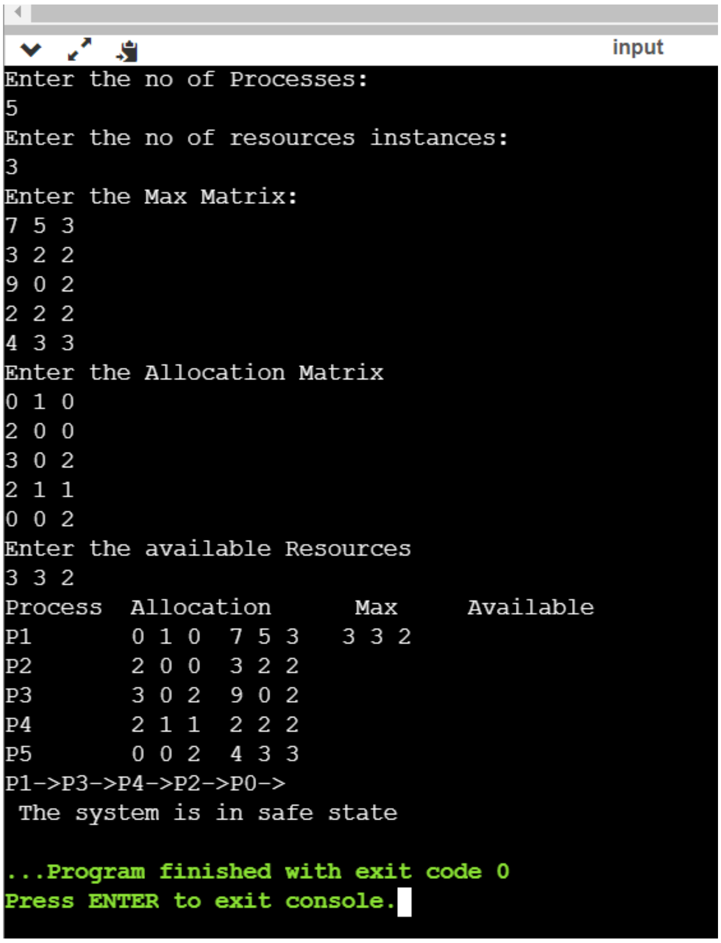
```
#include<stdio.h>
#include<conio.h>
int main()
{
        int max[100][100],alloc[100][100],need[100][100],avail[100];
        int finish[100],temp,safe[100],flag=1,k,c1=0;
  int n,r;
   int i,j;
        printf("Enter the no of Processes:\n");
        scanf("%d",&n);
        printf("Enter the no of resources instances:\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 Resources\n");
for(j=0;j<r;j++)
{
scanf("%d",&avail[j]);
}
printf("Process\t Allocation\t Max\t Available\t");
for(i=0;i<n;i++)
{
printf("\nP%d\t ",i+1);
        for(j=0;j<r;j++)
        {
        printf("%d ",alloc[i][j]);
        }
        printf("\t");
        for(j=0;j<r;j++)
        {
        printf("%d ",max[i][j]);
        }
        printf("\t");
        if(i==0)
        {
        for(j=0;j<r;j++)
                printf("%d ",avail[j]);
        }
}
for(i=0;i<n;i++)
{
finish[i]=0;
```

```
        }
//find need matrix
        for(i=0;i<n;i++)
        {
                for(j=0;j<r;j++)
                {
                need[i][j]=max[i][j]-alloc[i][j];
                }
        }
        printf("\n");
        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->",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("\n The system is in safe state");
}
else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
```

```
        }
}
```

Output:



```
Enter the no of Processes:
5
Enter the no of resources instances:
3
Enter the Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the Allocation Matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the available Resources
3 3 2
Process  Allocation      Max     Available
P1        0 1 0  7 5 3   3 3 2
P2        2 0 0  3 2 2
P3        3 0 2  9 0 2
P4        2 1 1  2 2 2
P5        0 0 2  4 3 3
P1->P3->P4->P2->P0->
 The system is in safe state

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

Aim: write a c program to simulate bankers algorithm for deadlock prevention

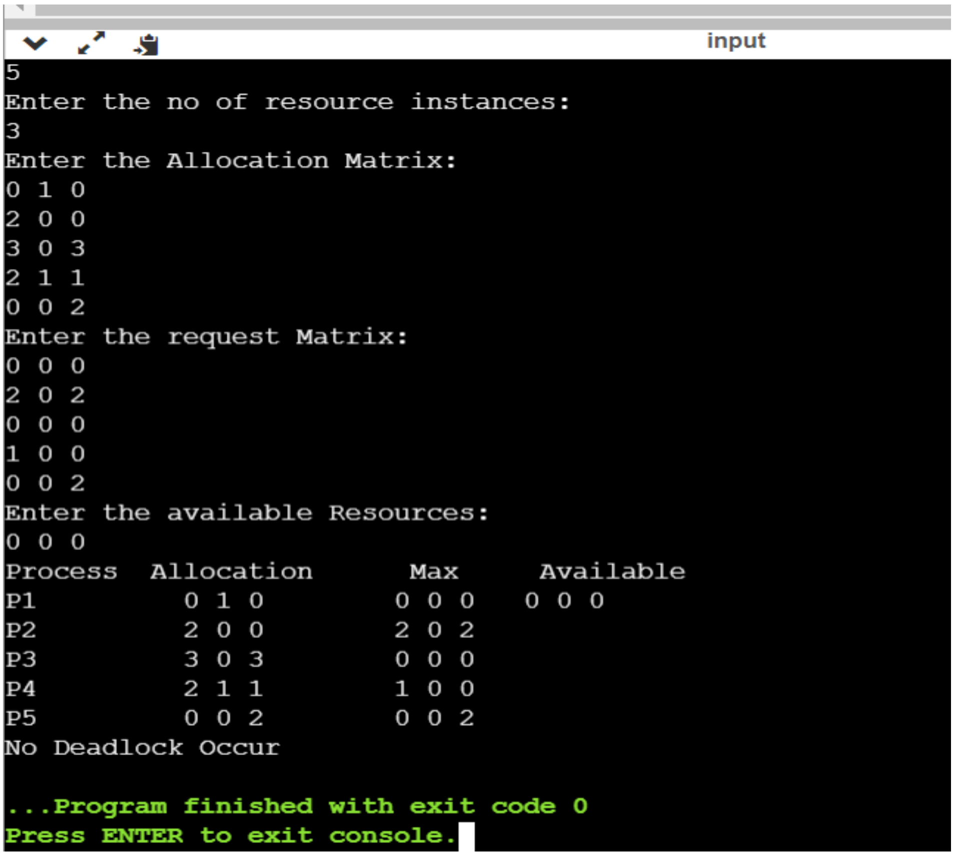
Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int max[100][100],alloc[100][100],need[100][100],avail[100];
   int finish[100],temp,flag=1,k,c1=0;
   int dead[100],safe[100];
   int n,r;
   int i,j;
   printf("Enter the no of Processes:\n");
   scanf("%d",&n);
   printf("Enter the no of resource instances:\n");
   scanf("%d",&r);
   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 request Matrix:\n");
   for(i=0;i<n;i++)
   {
      for(j=0;j<r;j++)
         {
            scanf("%d",&max[i][j]);
         }
   }
   printf("Enter the available Resources:\n");
   for(j=0;j<r;j++)
   {
      scanf("%d",&avail[j]);
   }
   printf("Process\t Allocation\t Max\t Available\t");
   for(i=0;i<n;i++)
   {
      printf("\nP%d\t   ",i+1);
      for(j=0;j<r;j++)
      {
         printf("%d ",alloc[i][j]);
      }
      printf("\t");
      for(j=0;j<r;j++)
      {
          printf("%d ",max[i][j]);
      }
      printf("\t");
         if(i==0)
         {
            for(j=0;j<r;j++)
            printf("%d ",avail[j]);
         }
   }
           for(i=0;i<n;i++)
           {
                   finish[i]=0;
           }
           //find need matrix
           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("\nP%d",i);
                            if(finish[i]==1)
                            {
                            i=n;
                            }
                        }
                    }
                }
     }
 }
 j=0;
 flag=0;
 for(i=0;i<n;i++)
 {
     if(finish[i]==0)
     {
         dead[j]=i;
         j++;
         flag=1;
     }
}
if(flag==1)
{
     printf("\n\nSystem is in Deadlock\n");
     //for(i=0;i<n;i++)
     //{
      //printf("P%d\t",dead[i]);
     //}
}
```

```
            else
            {
              printf("\nNo Deadlock Occur");
            }
}
```

Output:



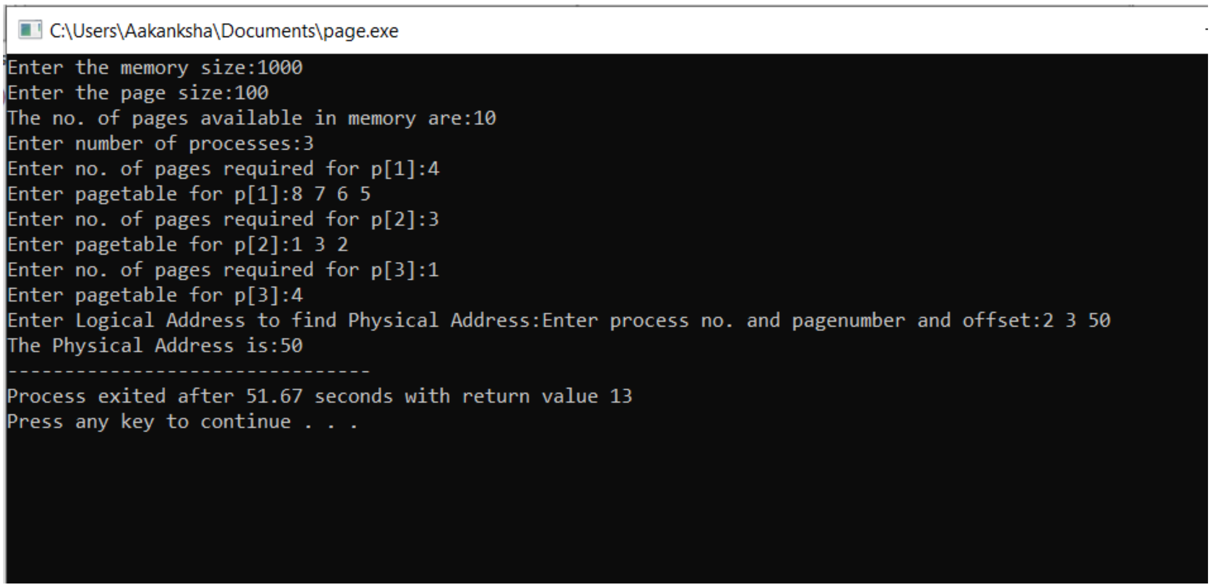
```
5
Enter the no of resource instances:
3
Enter the Allocation Matrix:
0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
Enter the request Matrix:
0 0 0
2 0 2
0 0 0
1 0 0
0 0 2
Enter the available Resources:
0 0 0
Process  Allocation      Max     Available
P1          0 1 0        0 0 0    0 0 0
P2          2 0 0        2 0 2
P3          3 0 3        0 0 0
P4          2 1 1        1 0 0
P5          0 0 2        0 0 2
No Deadlock Occur

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

Aim: write a c program to simulate paging technique of memory management

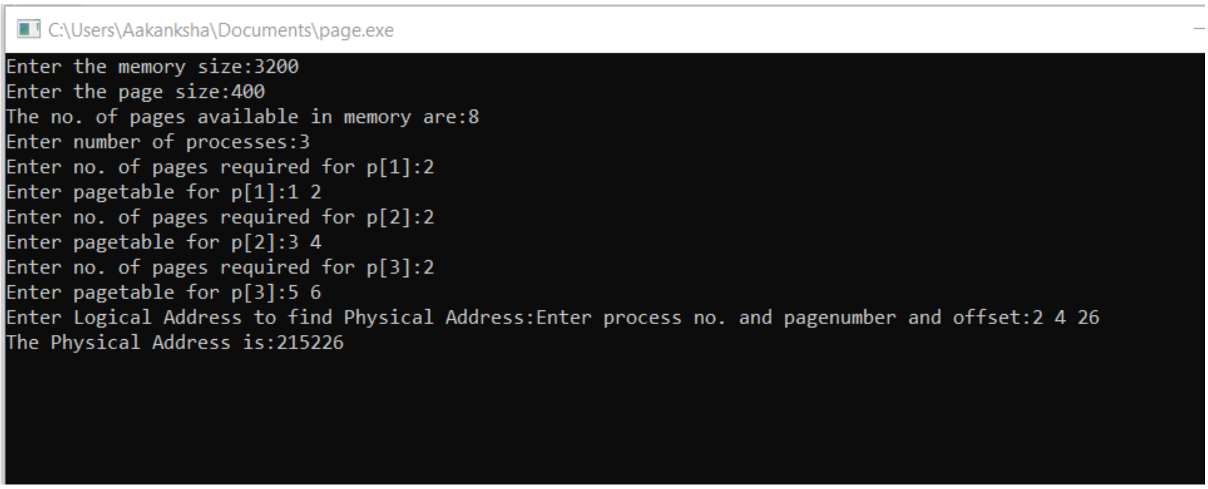
Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
        int s[10], fno[10][20];
        printf("Enter the memory size:");
        scanf("%d",&ms);
        printf("Enter the page size:");
        scanf("%d",&ps);
        nop = ms/ps;
        printf("The no. of pages available in memory are:%d ",nop);
        printf("Enter number of processes:");
        scanf("%d",&np);
        rempages = nop;
        for(i=1;i<=np;i++)
        {
                printf("Enter no. of pages required for p[%d]:",i);
                scanf("%d",&s[i]);
                if(s[i] >rempages)
                {
                printf("Memory is Full");
                break;
                }
                rempages = rempages - s[i];
                printf("Enter pagetable for p[%d]:",i);
                for(j=0;j<s[i];j++)
                scanf("%d",&fno[i][j]);
        }
        printf("Enter Logical Address to find Physical Address:");
        printf("Enter process no. and pagenumber and offset:");
        scanf("%d %d %d",&x,&y, &offset);
        if(x>np || y>=s[i] || offset>=ps)
        {
                printf("Invalid Process or Page Number or offset");
    }
        else
        {
        pa=fno[x][y]*ps+offset;
        printf("The Physical Address is:%d",pa);
        }
        getch();
}
```

Output:



```
Enter the memory size:1000
Enter the page size:100
The no. of pages available in memory are:10
Enter number of processes:3
Enter no. of pages required for p[1]:4
Enter pagetable for p[1]:8 7 6 5
Enter no. of pages required for p[2]:3
Enter pagetable for p[2]:1 3 2
Enter no. of pages required for p[3]:1
Enter pagetable for p[3]:4
Enter Logical Address to find Physical Address:Enter process no. and pagenumber and offset:2 3 50
The Physical Address is:50
--------------------------------
Process exited after 51.67 seconds with return value 13
Press any key to continue . . .
```



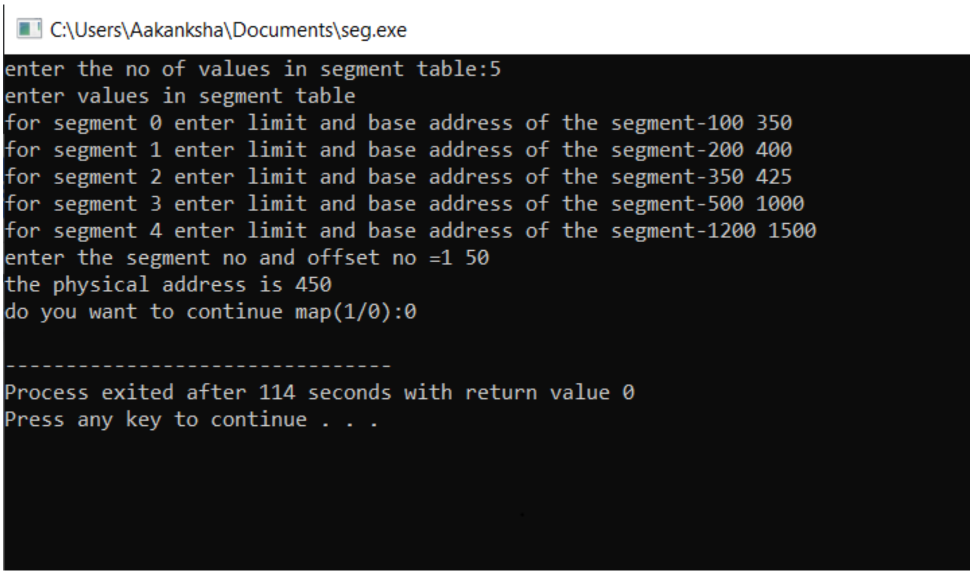
```
Enter the memory size:3200
Enter the page size:400
The no. of pages available in memory are:8
Enter number of processes:3
Enter no. of pages required for p[1]:2
Enter pagetable for p[1]:1 2
Enter no. of pages required for p[2]:2
Enter pagetable for p[2]:3 4
Enter no. of pages required for p[3]:2
Enter pagetable for p[3]:5 6
Enter Logical Address to find Physical Address:Enter process no. and pagenumber and offset:2 4 26
The Physical Address is:215226
```

Aim: write a c program to simulate segmentation technique of memory management

Program:

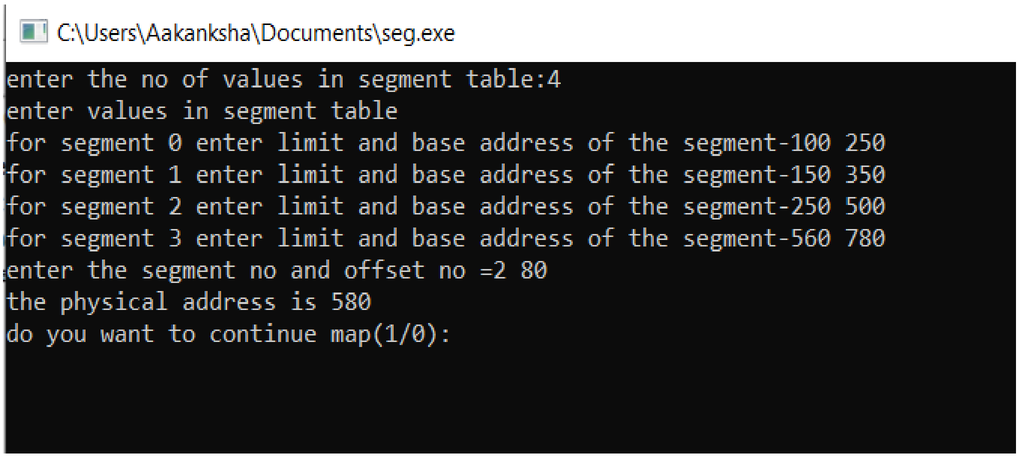
```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
struct seg_table
{
 int limit,base;
}st[10];
int verify_offset(int,int);
void main()
{
 int i,seg,d,n,val;
 int ch;
 printf("enter the no of values in segment table:");
 scanf("%d",&n);
 printf("enter values in segment table\n");
 for(i=0;i<n;i++)
 {
 printf("for segment %d enter limit and base address of the segment-",i);
  scanf("%d%d",&st[i].limit,&st[i].base);

}
do{
 printf("enter the segment no and offset no =");
 scanf("%d%d",&seg,&d);
 if(seg>=n){
 printf("segment no is exceeded\n");
 }
 else{
 val=verify_offset(seg,d);
 if(val!=0){
  printf("the physical address is %d\n",val);
 }
 }
 printf("do you want to continue map(1/0):");
 scanf("%d",&ch);
}
while(ch==1);
}
int verify_offset(int no,int off)
{
 int temp;
 if(off>st[no].limit)
 {
 printf("error, exceeding the memory\n");
 return 0;
 }
 else if(off<=st[no].limit)
 {
 temp=off+st[no].base;
 return(temp);
 }
}
```

Output:



C:\Users\Aakanksha\Documents\seg.exe

```
enter the no of values in segment table:5
enter values in segment table
for segment 0 enter limit and base address of the segment-100 350
for segment 1 enter limit and base address of the segment-200 400
for segment 2 enter limit and base address of the segment-350 425
for segment 3 enter limit and base address of the segment-500 1000
for segment 4 enter limit and base address of the segment-1200 1500
enter the segment no and offset no =1 50
the physical address is 450
do you want to continue map(1/0):0

--------------------------------
Process exited after 114 seconds with return value 0
Press any key to continue . . .
```



C:\Users\Aakanksha\Documents\seg.exe

```
enter the no of values in segment table:4
enter values in segment table
for segment 0 enter limit and base address of the segment-100 250
for segment 1 enter limit and base address of the segment-150 350
for segment 2 enter limit and base address of the segment-250 500
for segment 3 enter limit and base address of the segment-560 780
enter the segment no and offset no =2 80
the physical address is 580
do you want to continue map(1/0):
```

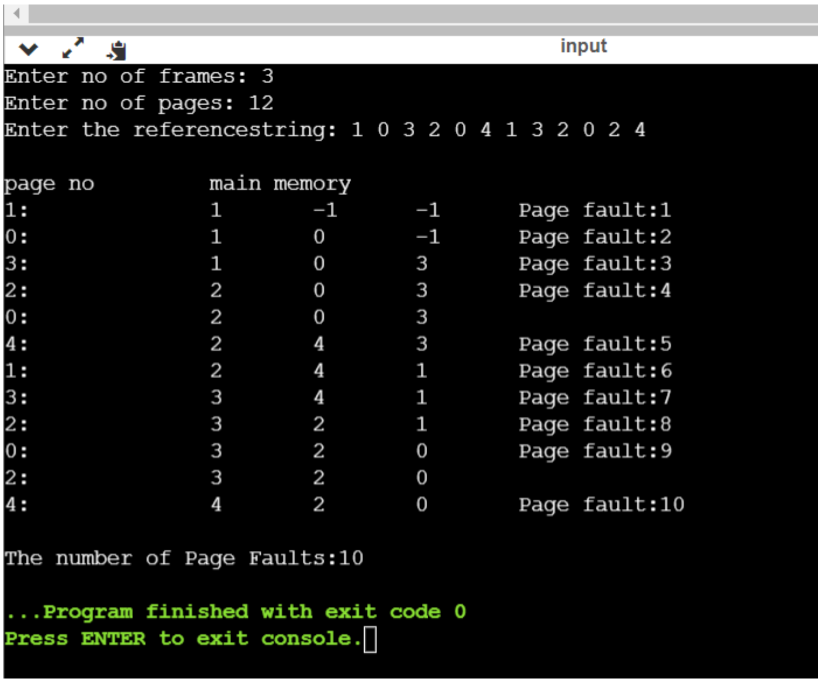
Aim: write a c program to simulate FIFO page replacement algorithm

Program:

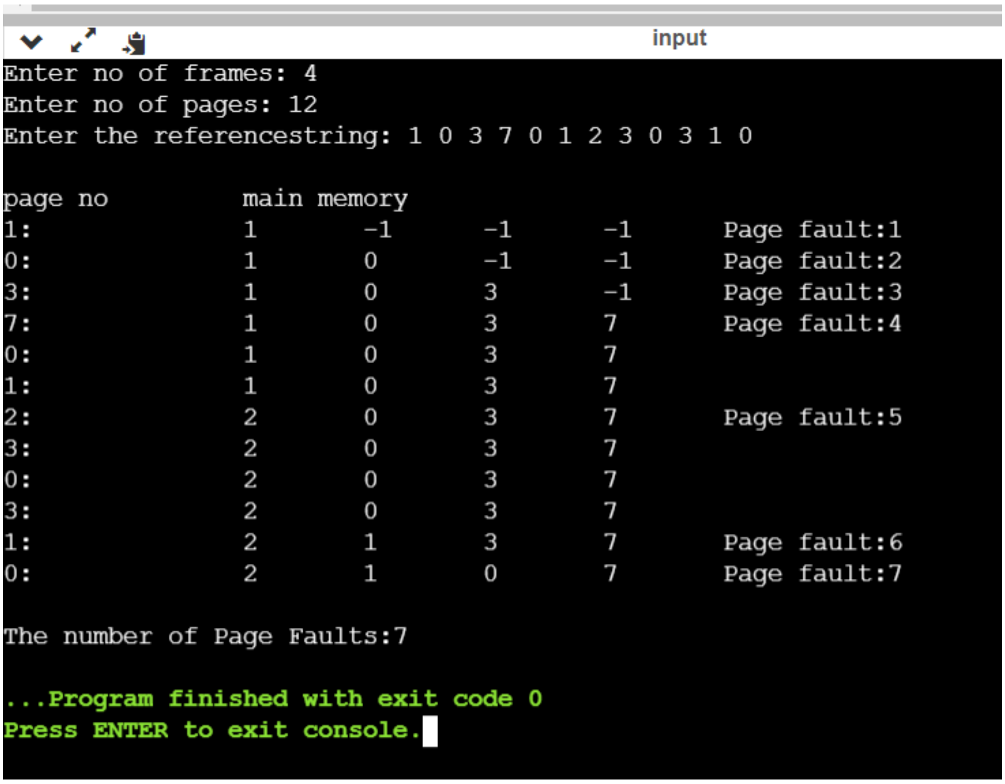
```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i,j,k,f,pf=0,count=0,rs[25],m[10],n;
   printf("Enter no of frames: ");
   scanf("%d",&f);
   printf("Enter no of pages: ");
   scanf("%d",&n);
   printf("Enter the referencestring: ");
   for(i=0;i<n;i++)
          scanf("%d",&rs[i]);
   for(i=0;i<f;i++)
          m[i]=-1;
   printf("\npage no\t\tmain memory\n");
   for(i=0;i<n;i++)
   {
      printf("%d:\t",rs[i]);
         for(k=0;k<f;k++)
         {
                    if(m[k]==rs[i]) //page is already in memory
                               break;
         }
```

```
        if(k==f)
        {
                m[count++]=rs[i];
                pf++;
        }
        for(j=0;j<f;j++)
                printf("\t%d",m[j]);
        if(k==f)
                printf("\tPage fault:%d",pf);
        printf("\n");
        if(count==f)
                count=0;
   }
   printf("\nThe number of Page Faults:%d",pf);
   getch();
}
```

Output:



```
Enter no of frames: 3
Enter no of pages: 12
Enter the referencestring: 1 0 3 2 0 4 1 3 2 0 2 4

page no         main memory
1:              1       -1      -1      Page fault:1
0:              1       0       -1      Page fault:2
3:              1       0       3       Page fault:3
2:              2       0       3       Page fault:4
0:              2       0       3
4:              2       4       3       Page fault:5
1:              2       4       1       Page fault:6
3:              3       4       1       Page fault:7
2:              3       2       1       Page fault:8
0:              3       2       0       Page fault:9
2:              3       2       0
4:              4       2       0       Page fault:10

The number of Page Faults:10

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



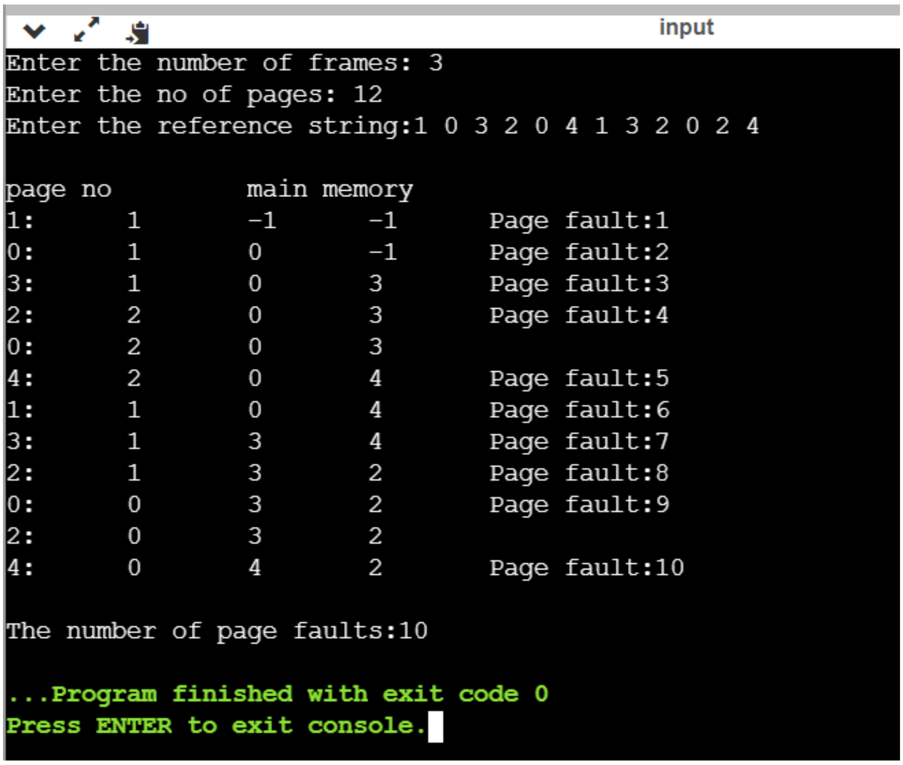
```
Enter no of frames: 4
Enter no of pages: 12
Enter the referencestring: 1 0 3 7 0 1 2 3 0 3 1 0

page no         main memory
1:              1       -1      -1      -1      Page fault:1
0:              1       0       -1      -1      Page fault:2
3:              1       0       3       -1      Page fault:3
7:              1       0       3       7       Page fault:4
0:              1       0       3       7
1:              1       0       3       7
2:              2       0       3       7       Page fault:5
3:              2       0       3       7
0:              2       0       3       7
3:              2       0       3       7
1:              2       1       3       7       Page fault:6
0:              2       1       0       7       Page fault:7

The number of Page Faults:7

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

Aim: write a c program to simulate LRU page replacement algorithm

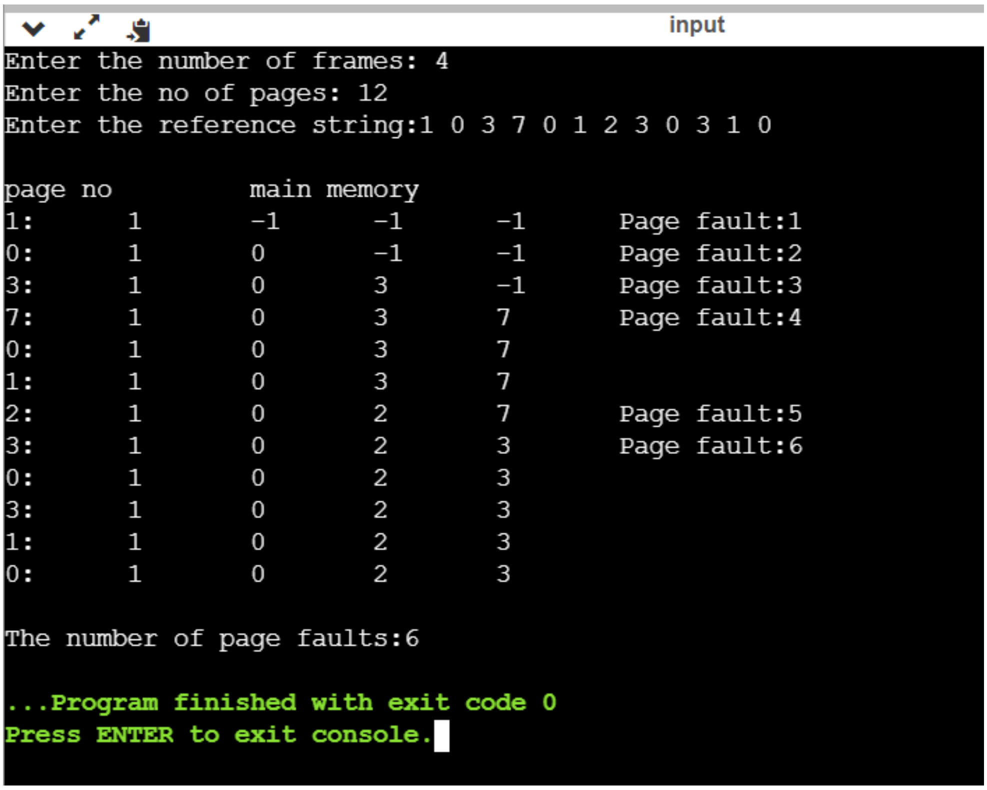
Program:

```
#include<stdio.h>
#include<conio.h>
void main(){
        int i,j,k,min,rs[25],m[10],count[10],flag[25],n,f,pf=0,next=1;
        printf("Enter the number of frames: ");
        scanf("%d",&f);
        printf("Enter the no of pages: ");
        scanf("%d",&n);
        printf("Enter the reference string:");
        for(i=0;i<n;i++)  {
        scanf("%d",&rs[i]);
        flag[i]=0;
        }
        for(i=0;i<f;i++)   {
                count[i]=0;
                m[i]=-1;
        }
        printf("\npage no\t\tmain memory\n");
        for(i=0;i<n;i++)  {
          printf("%d:\t",rs[i]);
        for(j=0;j<f;j++)              {
        if(m[j]==rs[i])               {
        flag[i]=1;
        count[j]=next;
        next++;
        }
        }
        if(flag[i]==0)      {
        if(i<f)             {
        m[i]=rs[i];
        count[i]=next;
        next++;
        }
        else      {
        min=0;
        for(j=1;j<f;j++)
        if(count[min]>count[j])min=j;
        m[min]=rs[i];
        count[min]=next;
        next++;
        }
        pf++;
        }
        for(j=0;j<f;j++)
           printf("%d\t", m[j]);
        if(flag[i]==0)
        printf("Page fault:%d" ,pf);
        printf("\n");
        }
        printf("\nThe number of page faults:%d",pf);
        getch();
}
```

Output:



```
Enter the number of frames: 3
Enter the no of pages: 12
Enter the reference string:1 0 3 2 0 4 1 3 2 0 2 4

page no         main memory
1:       1       -1       -1       Page fault:1
0:       1       0        -1       Page fault:2
3:       1       0        3        Page fault:3
2:       2       0        3        Page fault:4
0:       2       0        3
4:       2       0        4        Page fault:5
1:       1       0        4        Page fault:6
3:       1       3        4        Page fault:7
2:       1       3        2        Page fault:8
0:       0       3        2        Page fault:9
2:       0       3        2
4:       0       4        2        Page fault:10

The number of page faults:10

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



```
Enter the number of frames: 4
Enter the no of pages: 12
Enter the reference string:1 0 3 7 0 1 2 3 0 3 1 0

page no         main memory
1:       1       -1       -1       -1       Page fault:1
0:       1       0        -1       -1       Page fault:2
3:       1       0        3        -1       Page fault:3
7:       1       0        3        7        Page fault:4
0:       1       0        3        7
1:       1       0        3        7
2:       1       0        2        7        Page fault:5
3:       1       0        2        3        Page fault:6
0:       1       0        2        3
3:       1       0        2        3
1:       1       0        2        3
0:       1       0        2        3

The number of page faults:6

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