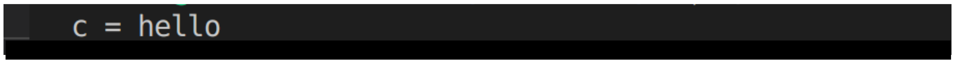
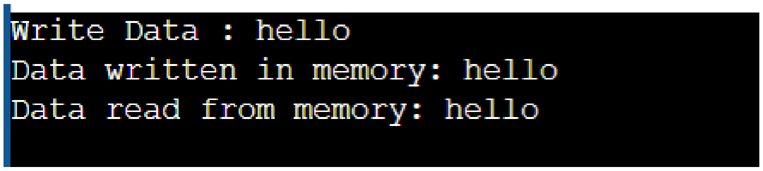
```
// system Call

#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<stdlib.h>
#include <string.h>

int main()
{
    int fd[2];
    char c[10];
    char buf1[12] = "hello world";
    fd[0] = open("hello.txt", O_RDWR);
    fd[1] = open("h.txt", O_RDWR);

    write(fd[0], buf1, strlen(buf1));

    read(fd[1], &c, 5);

    printf("c = %s\n", c);
    close(fd[0]);
    close(fd[1]);

}
```



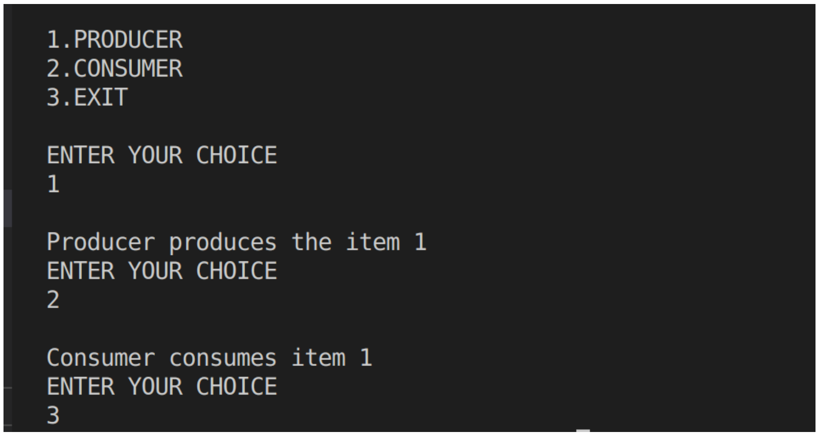
```
c = hello
```

```
//IPC using shared memory


#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);

    printf("Write Data : ");
    scanf("%[^\n]%*c", str);

    printf("Data written in memory: %s\n",str);

    //detach from shared memory
    shmdt(str);

    // shmat to attach to shared memory
    char *s = (char*) shmat(shmid,(void*)0,0);

    printf("Data read from memory: %s\n",s);

    //detach from shared memory
    shmdt(s);

    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL);

    return 0;
}
```



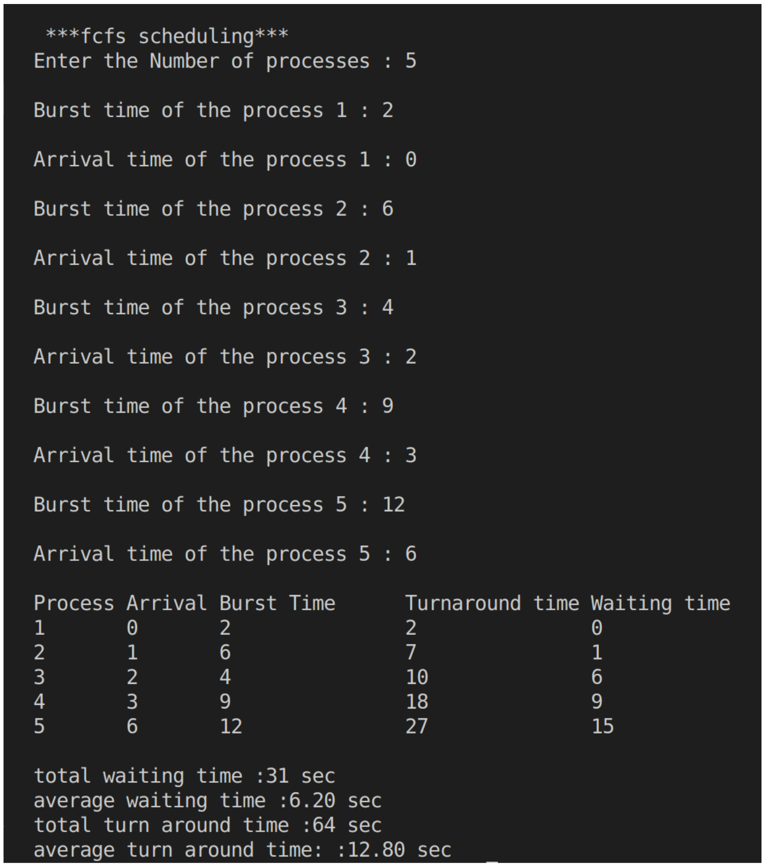
```
Write Data : hello
Data written in memory: hello
Data read from memory: hello
```

```
//Semaphore

#include<stdio.h>
#include <stdlib.h>
int mutex=1,full=0,empty=3,x=0;

void producer();
void consumer();
int wait(int);
int signal(int);

void main()
{
int n;
printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
while(1) {
printf("\nENTER YOUR CHOICE\n");
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;
}
}
}

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);
}
```



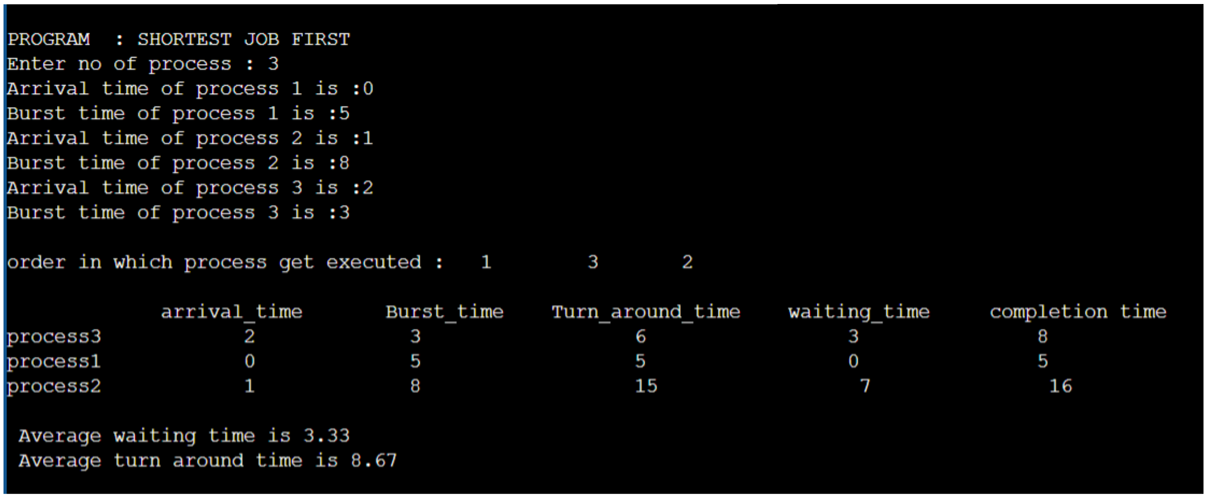
```
1.PRODUCER
2.CONSUMER
3.EXIT

ENTER YOUR CHOICE
1

Producer produces the item 1
ENTER YOUR CHOICE
2

Consumer consumes item 1
ENTER YOUR CHOICE
3
```

```
//FCFS


#include <stdio.h>
#include <stdlib.h>

#define SIZE 1000

struct sjf{
int pid;
int btime;
int wtime;
int ttime;
int arrtime;
int comptime;
} p[10];

void sort(struct sjf *p,int n){
for (int i = 0;i<n;i++){
for (int j=0;j<n-1-i;j++){
if (p[j].arrtime>p[j+1].arrtime){
struct sjf temp = p[j];
p[j] = p[j+1];
p[j+1] = temp;
}
}
}
}

int main(){
int i,n;
int towtwtime=0,totttime=0;
printf("\n ***fcfs scheduling***\n");
printf("Enter the Number of processes : ");
scanf("%d",&n);
for (i=0;i<n;i++){
p[i].pid=i+1;
printf("\nBurst time of the process %d : ",i+1);
scanf("%d",&p[i].btime);
printf("\nArrival time of the process %d : ",i+1);
scanf("%d",&p[i].arrtime);
}

sort(p,n);

p[0].wtime=0;
p[0].ttime=p[0].comptime=p[0].btime;
totttime+=p[0].ttime;
for(i=1;i<n;i++)
{
p[i].wtime=(p[i].arrtime-p[i-1].comptime>=0)?0:(p[i-1].comptime-p[i].arrtime);
p[i].ttime=p[i].wtime+p[i].btime;
p[i].comptime = p[i].arrtime+p[i].wtime+p[i].btime;
totttime+=p[i].ttime;
towtwtime+=p[i].wtime;
}

printf("\nProcess\tArrival\tBurst Time\tTurnaround time\tWaiting time\n");
for(i=0;i<n;i++)
{
 printf("%d\t",p[i].pid);
 printf("%d\t",p[i].arrtime);
 printf("%d\t\t",p[i].btime);
 printf("%d\t\t",p[i].ttime);
 printf("%d\n",p[i].wtime);
}

printf("\ntotal waiting time :%d sec", towtwtime );
printf("\naverage waiting time :%.2f sec",(float)towtwtime/n);
printf("\ntotal turn around time :%d sec",totttime);
printf("\naverage turn around time: :%.2f sec\n",(float)totttime/n);
}
```



```
 ***fcfs scheduling***
Enter the Number of processes : 5

Burst time of the process 1 : 2

Arrival time of the process 1 : 0

Burst time of the process 2 : 6

Arrival time of the process 2 : 1

Burst time of the process 3 : 4

Arrival time of the process 3 : 2

Burst time of the process 4 : 9

Arrival time of the process 4 : 3

Burst time of the process 5 : 12

Arrival time of the process 5 : 6

Process Arrival Burst Time      Turnaround time Waiting time
1       0       2               2               0
2       1       6               7               1
3       2       4               10              6
4       3       9               18              9
5       6       12              27              15

total waiting time :31 sec
average waiting time :6.20 sec
total turn around time :64 sec
average turn around time: :12.80 sec
```

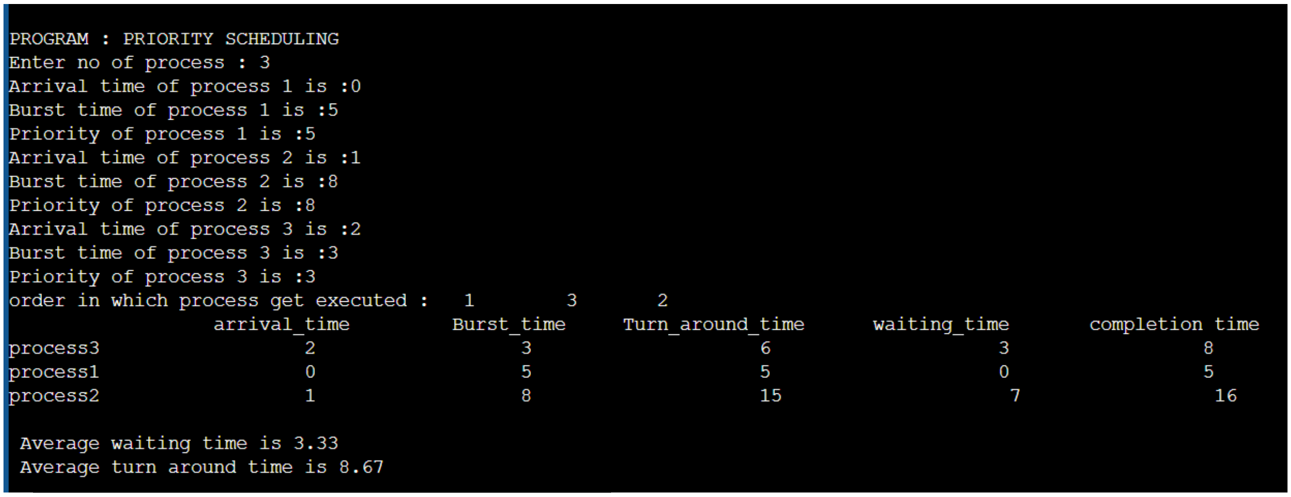
```
//SJF

#include <stdio.h>
#define SIZE 1000

typedef struct {
    int pid;
    int wtime;
    int tatime;
    int btime;
    int arrtime;
    int comptime;
    int si;
} process;

void sort(process *p,int n){
            for (int i = 0;i<n;i++){
                        for (int j=0;j<n-1-i;j++){
                                    if (p[j].arrtime>p[j+1].arrtime){
                                                process temp = p[j];
                                                p[j] = p[j+1];
                                                p[j+1] = temp;
                                    }

                        }
            }
}

process heap[SIZE];
int tail = -1;

void insert(process p){
    if (tail+1>=SIZE){
        return;
    }

    heap[++tail] = p;


    //heapify-up
    int childIndex = tail;
    int parentIndex = (tail-1)/2;

    while (childIndex>0 && heap[parentIndex].btime>heap[childIndex].btime){

        process temp = heap[parentIndex];
        heap[parentIndex] = heap[childIndex];
        heap[childIndex] = temp;
        childIndex = parentIndex;
        parentIndex = (childIndex-1)/2;
    }
}

int isempty(){
    return (tail<=-1);
}

process delete(){
    if (tail==-1){
        return;
    }

    process temp = heap[0];
    heap[0] = heap[tail--];

    //heapify-down
    int parentIndex = 0;
    int leftChildIndex = 2*parentIndex+1;
    int rightChildIndex = 2*parentIndex+2;

    while (leftChildIndex<=tail){
        int minIndex =
(heap[parentIndex].btime>heap[leftChildIndex].btime)?leftChildIndex:parentIndex;

        if (rightChildIndex<=tail){
            minIndex =
(heap[minIndex].btime>heap[rightChildIndex].btime)?rightChildIndex:minIndex;
        }

        if (minIndex==parentIndex){
            break;
        }

        process temp = heap[parentIndex];
        heap[parentIndex] = heap[minIndex];
        heap[minIndex] = temp;

        parentIndex = minIndex;
        leftChildIndex = 2*parentIndex+1;
        rightChildIndex = 2*parentIndex+2;
    }

    return temp;
}

void sjf(process *p,int n){
    sort(p,n);
    for (int i = 0;i<n;i++){
            p[i].si = i;
    }
    int time = p[0].arrtime;
    int cp = 0;
    while (cp<n||!isempty()){
        while (cp<n && p[cp].arrtime<=time){

            insert(p[cp++]);
        }
        process temp = delete();

        p[temp.si].wtime = time-temp.arrtime;
        p[temp.si].tatime = p[temp.si].wtime+p[temp.si].btime;
        time+=temp.btime;
        p[temp.si].comptime = time;

    }
}

void main(){
    int n;
    printf("Enter the number of processes : ");
    scanf("%d",&n);

    process p[SIZE];
    for (int i=0;i<n;i++){
        printf("Enter arrival time : ");
        scanf("%d",&p[i].arrtime);

        printf("Enter burst time : ");
        scanf("%d",&p[i].btime);

        p[i].pid = i;
    }
    printf("\n");
    sjf(p,n);

    int totalWT = 0,totalTT = 0;
    for (int i = 0;i<n;i++){
        printf("waiting time of process %d : %d\n",p[i].pid+1,p[i].wtime);
            totalWT+=p[i].wtime;
            totalTT+=p[i].tatime;
        printf("turnaround time of process %d : %d\n",p[i].pid+1,p[i].tatime);
        printf("completion time of process %d : %d\n\n",p[i].pid+1,p[i].comptime);

    }
    printf("\nTotal waiting time : %d",totalWT);
    printf("\nTotal turn around time : %d",totalTT);
    printf("\nAverage waiting time : %f",(float)totalWT/n);
    printf("\nAverage turn around time : %f\n\n",(float)totalTT/n);

}
```



```
PROGRAM  : SHORTEST JOB FIRST
Enter no of process : 3
Arrival time of process 1 is :0
Burst time of process 1 is :5
Arrival time of process 2 is :1
Burst time of process 2 is :8
Arrival time of process 3 is :2
Burst time of process 3 is :3

order in which process get executed :   1        3       2

             arrival_time       Burst_time    Turn_around_time    waiting_time     completion time
process3            2              3                  6                 3               8
process1            0              5                  5                 0               5
process2            1              8                  15                 7               16

 Average waiting time is 3.33
 Average turn around time is 8.67
```

```
//Priority

#include <stdio.h>
#define SIZE 1000

typedef struct {
    int pid;
    int wtime;
    int tatime;
    int btime;
    int arrtime;
    int comptime;
    int priority;
    int si;
} process;

void sort(process *p,int n){
            for (int i = 0;i<n;i++){
                        for (int j=0;j<n-1-i;j++){
                                    if (p[j].arrtime>p[j+1].arrtime){
                                                process temp = p[j];
                                                p[j] = p[j+1];
                                                p[j+1] = temp;
                                    }

                        }
            }
}

process heap[SIZE];
int tail = -1;

void insert(process p){
    if (tail+1>=SIZE){
        return;
    }

    heap[++tail] = p;


    //heapify-up
    int childIndex = tail;
    int parentIndex = (tail-1)/2;

    while (childIndex>0 && heap[parentIndex].priority>heap[childIndex].priority){

        process temp = heap[parentIndex];
        heap[parentIndex] = heap[childIndex];
        heap[childIndex] = temp;
        childIndex = parentIndex;
        parentIndex = (childIndex-1)/2;
    }



}int totalWT = 0,totalTT = 0;

int isempty(){
    return (tail==-1);
}

process delete(){
    if (tail==-1){
        return;
    }

    process temp = heap[0];
    heap[0] = heap[tail--];

    //heapify-down
    int parentIndex = 0;
    int leftChildIndex = 2*parentIndex+1;
    int rightChildIndex = 2*parentIndex+2;

    while (leftChildIndex<=tail){
        int minIndex =
(heap[parentIndex].btime>heap[leftChildIndex].btime)?leftChildIndex:parentIndex;

        if (rightChildIndex<=tail){
            minIndex =
(heap[minIndex].btime>heap[rightChildIndex].btime)?rightChildIndex:minIndex;
        }

        if (minIndex==parentIndex){
            break;
        }

        process temp = heap[parentIndex];
        heap[parentIndex] = heap[minIndex];
        heap[minIndex] = temp;

        parentIndex = minIndex;
        leftChildIndex = 2*parentIndex+1;
        rightChildIndex = 2*parentIndex+2;
    }

    return temp;
```

```
}

void priority(process *p,int n){
    //lower priority no highest priority
    sort(p,n);

    for (int i = 0;i<n;i++){
            p[i].si = i;
    }
    int time = p[0].arrtime;
    int cp = 0;
    while (cp<n||!isempty()){
        while (cp<n && p[cp].arrtime<=time){
            insert(p[cp++]);
        }

        if (isempty()){
            time = p[cp].arrtime;

            continue;
        }

        process temp = delete();

        p[temp.si].wtime = time-temp.arrtime;
        p[temp.si].tatime = p[temp.si].wtime+temp.btime;
        time+=temp.btime;
            p[temp.si].comptime = time;
    }
}

void main(){
    int n;
    printf("Enter the number of processes : ");
    scanf("%d",&n);

    process p[SIZE];
    for (int i=0;i<n;i++){
        printf("Enter arrival time : ");
        scanf("%d",&p[i].arrtime);

        printf("Enter burst time : ");
        scanf("%d",&p[i].btime);

        printf("Enter priority : ");
        scanf("%d",&p[i].priority);
        p[i].pid = i;
    }
    printf("\n");
    priority(p,n);

    int totalWT = 0,totalTT = 0;
    for (int i = 0;i<n;i++){
        printf("waiting time of process %d : %d\n",p[i].pid+1,p[i].wtime);
                totalWT+=p[i].wtime;
                totalTT+=p[i].tatime;
                printf("completion time of process %d : %d\n",p[i].pid+1,p[i].comptime);
        printf("turnaround time of process %d : %d\n\n",p[i].pid+1,p[i].tatime);
    }

    printf("\nTotal waiting time : %d",totalWT);
    printf("\nTotal turn around time : %d",totalTT);
    printf("\nAverage waiting time : %f",(float)totalWT/n);
    printf("\nAverage turn around time : %f\n\n",(float)totalTT/n);

}
```

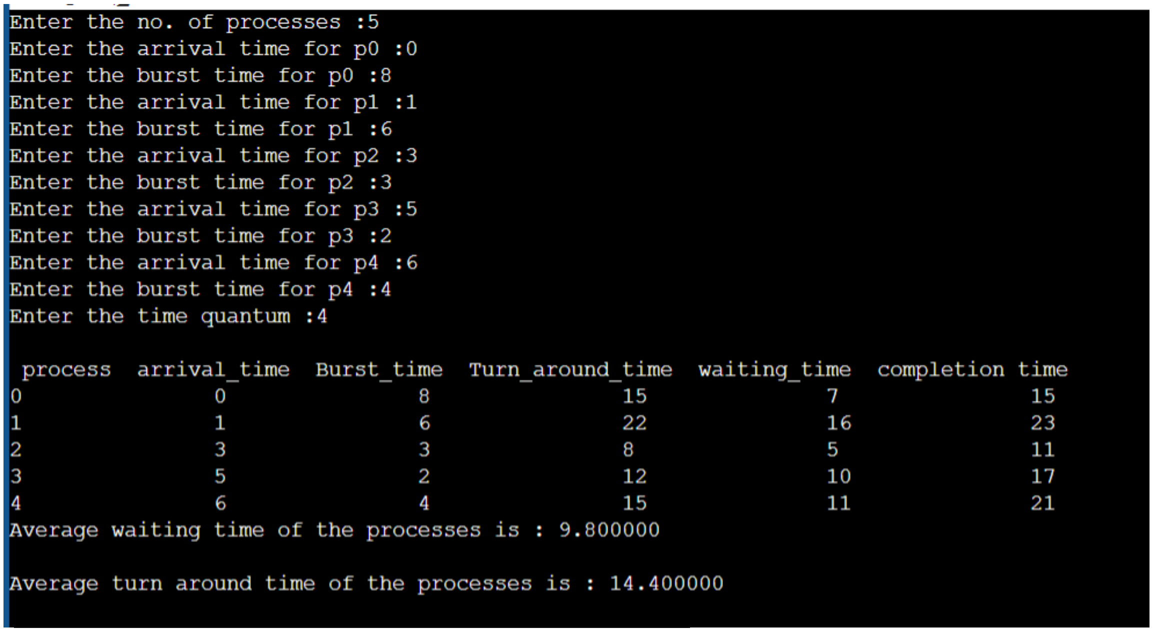


```
PROGRAM : PRIORITY SCHEDULING
Enter no of process : 3
Arrival time of process 1 is :0
Burst time of process 1 is :5
Priority of process 1 is :5
Arrival time of process 2 is :1
Burst time of process 2 is :8
Priority of process 2 is :8
Arrival time of process 3 is :2
Burst time of process 3 is :3
Priority of process 3 is :3
order in which process get executed :   1        3       2
                  arrival_time         Burst_time     Turn_around_time      waiting_time       completion time
process3                  2                  3                    6                    3                 8
process1                  0                  5                    5                    0                 5
process2                  1                  8                    15                    7                 16

 Average waiting time is 3.33
 Average turn around time is 8.67
```

```
//RR


#include <stdio.h>
#define SIZE 1000

typedef struct process{
    int pid;
    int wtime;
    int tatime;
    int btime;
    int arrtime;
    int comptime;
    int btime1;
    // int priority;
    struct process *next;
} process;

void sort(process *p,int n){
            for (int i = 0;i<n;i++){
                        for (int j=0;j<n-1-i;j++){
                                    if (p[j].arrtime>p[j+1].arrtime){
                                                process temp = p[j];
                                                p[j] = p[j+1];
                                                p[j+1] = temp;
                                    }

                        }
            }
}



process *head,*tail;

int isempty(){
    return (!head);
}

void enqueue(process *p){
    if (!head){
        head = tail = p;
        // tail->next = NULL;
        return;
    }

    tail->next = p;
    tail = tail->next;
}
```

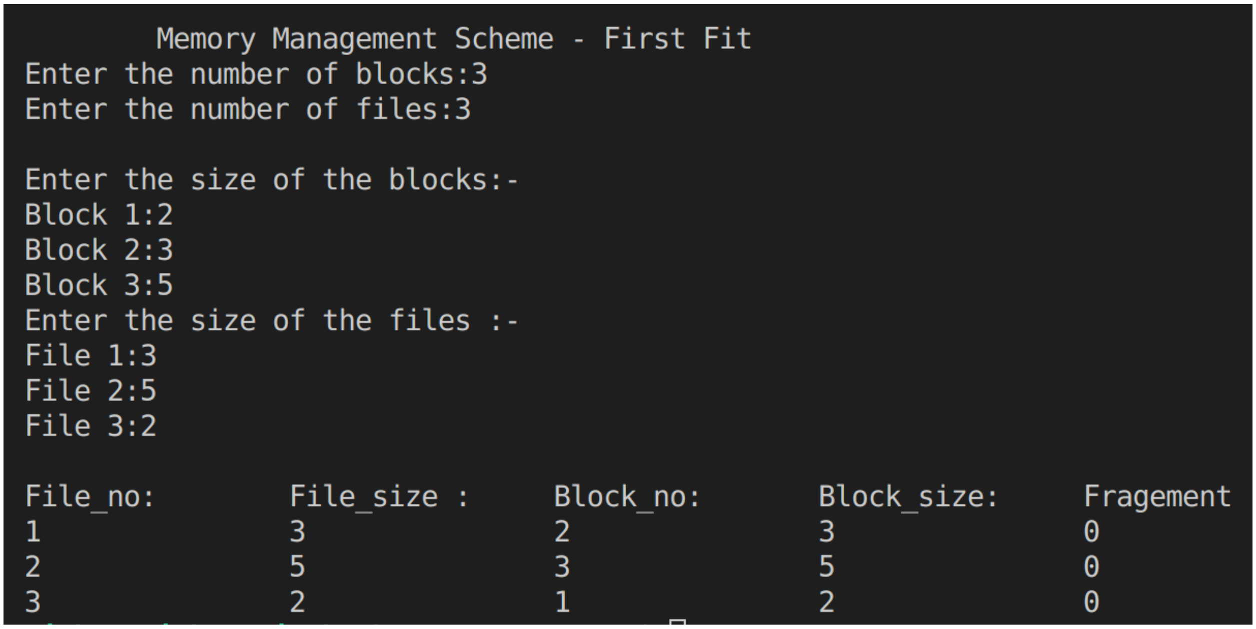
```
process * dequeue(){
    if (!head){
        return NULL;
    }

    process *temp = head;
    if (head==tail){
        head=tail=NULL;
    } else {
        head = head->next;

    }
    temp->next = NULL;

    return temp;
}

void rr(process *p,int n,int t){
    //lower priority no highest priority
    sort(p,n);
    int time = p[0].arrtime;
    int cp = 0;
    while (cp<n||!isempty()){
        while (cp<n && p[cp].arrtime<=time){

            enqueue(&p[cp++]);
        }



        process *temp = dequeue();


        int ta = (temp->btime1>=t)?t:temp->btime1;
        temp->btime1-=ta;
        time+=ta;

        if (temp->btime1<=0){

            temp->comptime=time;
            temp->wtime = time-temp->btime-temp->arrtime;
            temp->tatime = time-temp->arrtime;


            while (cp<n && p[cp].arrtime<=time){

                enqueue(&p[cp++]);
            }
        } else {
            while (cp<n && p[cp].arrtime<=time){

                enqueue(&p[cp++]);
            }


            enqueue(temp);
        }
    }
}

void main(){
    int n,t;

    printf("Enter the time slice : ");
    scanf("%d",&t);
    printf("Enter the number of processes : ");
    scanf("%d",&n);

    process p[SIZE];
    for (int i=0;i<n;i++){
        printf("Enter arrival time : ");
        scanf("%d",&p[i].arrtime);

        printf("Enter burst time : ");
        scanf("%d",&p[i].btime);
        p[i].btime1 = p[i].btime;

        p[i].pid = i;
    }
    printf("\n");
    rr(p,n,t);

    int totalWT = 0,totalTT = 0;
    for (int i = 0;i<n;i++){
        printf("waiting time of process %d : %d\n",p[i].pid+1,p[i].wtime);
            totalWT+=p[i].wtime;
            totalTT+=p[i].tatime;
        printf("turnaround time of process %d : %d\n",p[i].pid+1,p[i].tatime);
            printf("completion time of process %d : %d\n\n",p[i].pid+1,p[i].comptime);
    }

    printf("\nTotal waiting time : %d",totalWT);
    printf("\nTotal turn around time : %d",totalTT);
    printf("\nAverage waiting time : %f",(float)totalWT/n);
    printf("\nAverage turn around time : %f\n\n",(float)totalTT/n);
```

```
}
```



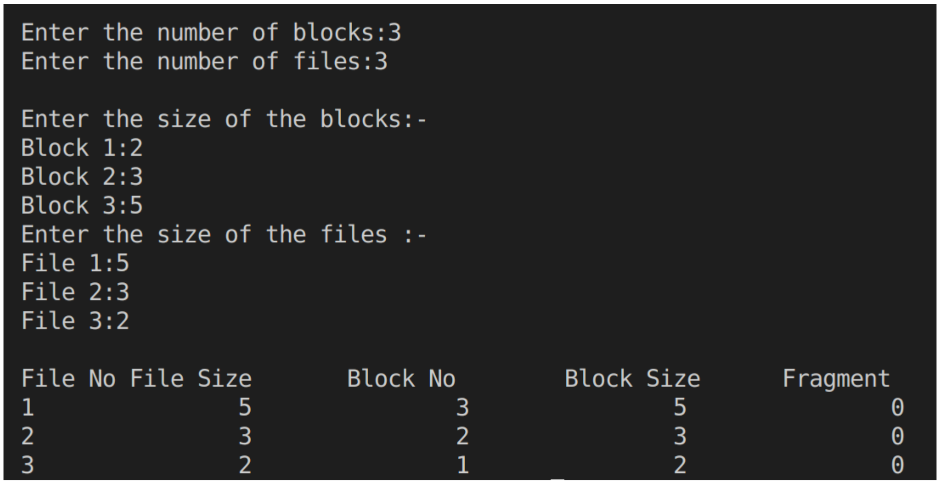
```
Enter the no. of processes :5
Enter the arrival time for p0 :0
Enter the burst time for p0 :8
Enter the arrival time for p1 :1
Enter the burst time for p1 :6
Enter the arrival time for p2 :3
Enter the burst time for p2 :3
Enter the arrival time for p3 :5
Enter the burst time for p3 :2
Enter the arrival time for p4 :6
Enter the burst time for p4 :4
Enter the time quantum :4

 process  arrival_time  Burst_time  Turn_around_time  waiting_time  completion time
0               0               8               15              7               15
1               1               6               22              16              23
2               3               3               8               5               11
3               5               2               12              10              17
4               6               4               15              11              21
Average waiting time of the processes is : 9.800000

Average turn around time of the processes is : 14.400000
```

```
//First fit


#include<stdio.h>
#include<curses.h>
#include <stdlib.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
system("clear");
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter 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("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
printf("\n");
getch();
}
```



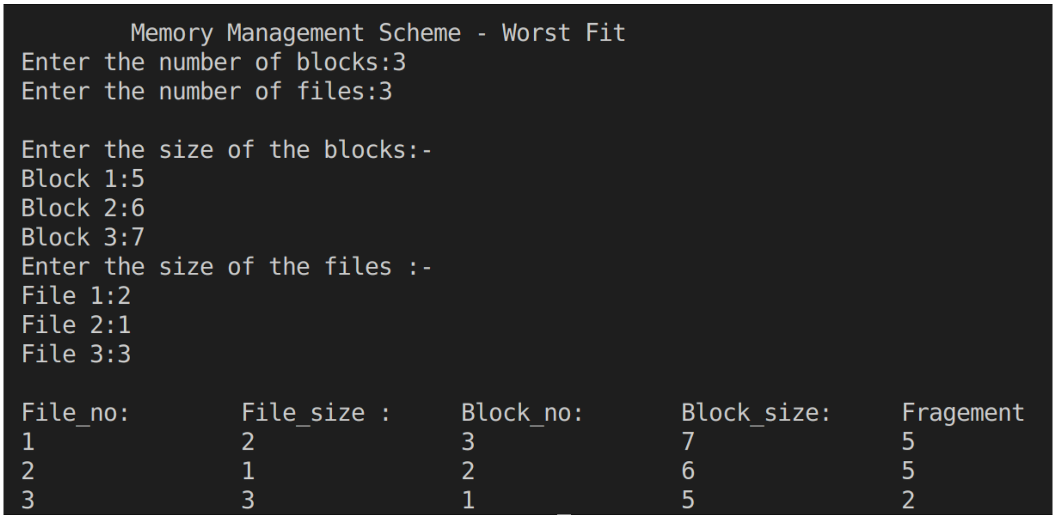
```
        Memory Management Scheme - First Fit
Enter the number of blocks:3
Enter the number of files:3

Enter the size of the blocks:-
Block 1:2
Block 2:3
Block 3:5
Enter the size of the files :-
File 1:3
File 2:5
File 3:2

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               3               2               3               0
2               5               3               5               0
3               2               1               2               0
```

```
//best fit
#include<stdio.h>
#include<curses.h>
#include <stdlib.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];
system("clear");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter 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("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
printf("\n");
}
```



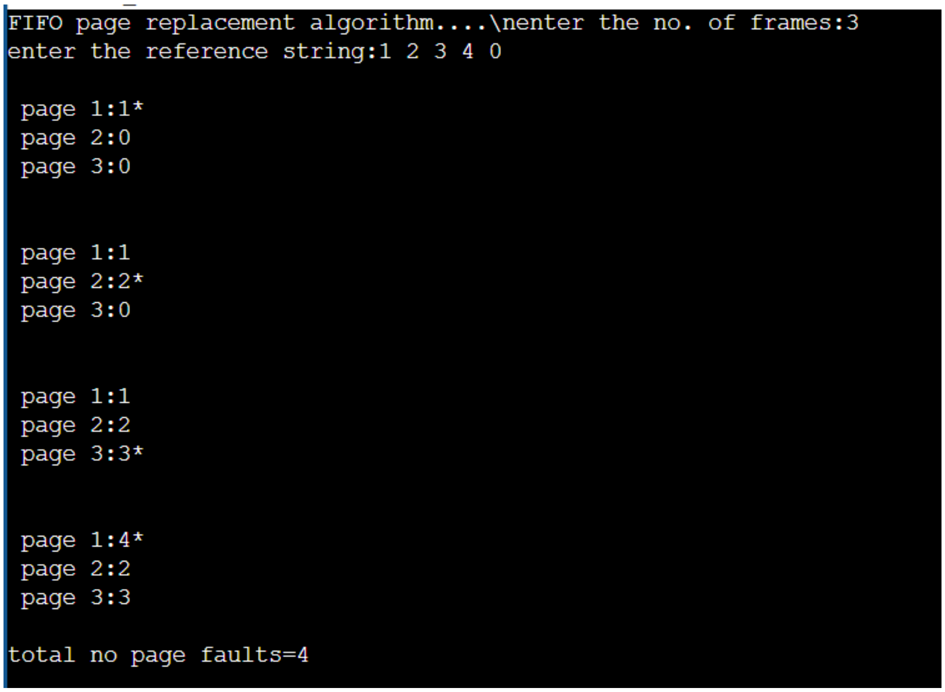
```
Enter the number of blocks:3
Enter the number of files:3

Enter the size of the blocks:-
Block 1:2
Block 2:3
Block 3:5
Enter the size of the files :-
File 1:5
File 2:3
File 3:2

File No File Size       Block No        Block Size      Fragment
1               5               3               5               0
2               3               2               3               0
3               2               1               2               0
```

```
//Worst fit

#include<stdio.h>
#include <curses.h>
#include <stdlib.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
system("clear");
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter 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) //if bf[j] is not allocated
{
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("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
printf("\n");
getch();
}
```



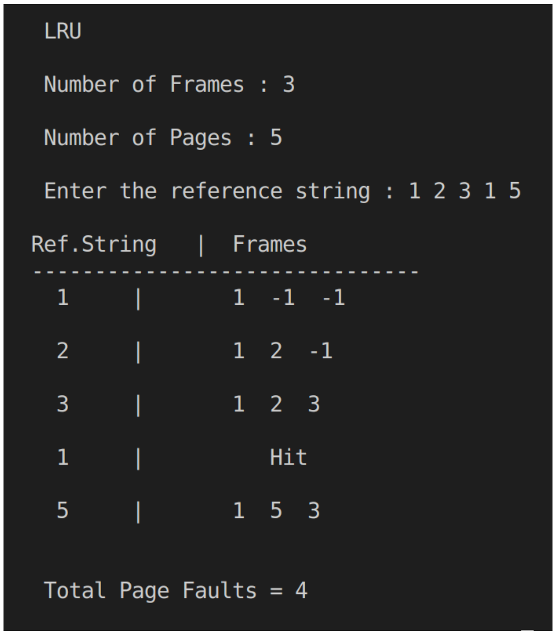
```
        Memory Management Scheme - Worst Fit
Enter the number of blocks:3
Enter the number of files:3

Enter the size of the blocks:-
Block 1:5
Block 2:6
Block 3:7
Enter the size of the files :-
File 1:2
File 2:1
File 3:3

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               2               3               7               5
2               1               2               6               5
3               3               1               5               2
```

```
//FIFO page replacement

#include<stdio.h>
#include<stdlib.h>
int main()
{
int i=0,j=0,k=0,i1=0,m,n,rs[30],flag=1,p[30];
system("clear");
printf("FIFO page replacement algorithm....\\n");
printf("enter the no. of frames:");
scanf("%d",&n);
printf("enter the reference string:");
while(1)
{
scanf("%d",&rs[i]);
if(rs[i]==0)
break;
i++;
}
m=i;
for(j=0;j<n;j++)
p[j]=0;
for(i=0;i<m;i++)
{
flag=1;
for(j=0;j<n;j++)
if(p[j]==rs[i])
{
printf("data already in page....\n");
flag=0;
break;
}
if(flag==1)
{
p[i1]=rs[i];
i1++;
k++;
if(i1==n)
i1=0;
for(j=0;j<n;j++)
{
printf("\n page %d:%d",j+1,p[j]);
if(p[j]==rs[i])
printf("*");
}
printf("\n\n");
}
}
printf("total no page faults=%d",k);
}
```



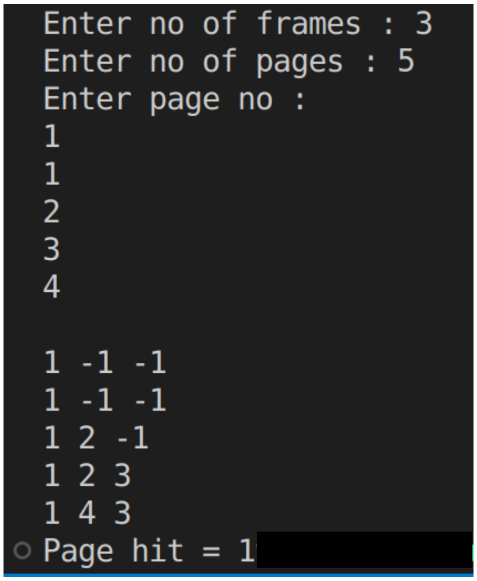
```
FIFO page replacement algorithm....\nenter the no. of frames:3
enter the reference string:1 2 3 4 0

 page 1:1*
 page 2:0
 page 3:0


 page 1:1
 page 2:2*
 page 3:0


 page 1:1
 page 2:2
 page 3:3*


 page 1:4*
 page 2:2
 page 3:3

total no page faults=4
```

```c
// LRU
#include <stdio.h>

int findLRU(int time[], int fCount) {
    int k, min, pos;
    pos = 0;
    min = time[0];
    for (k = 1; k < fCount; ++k) {
        if (time[k] < min) {
            min = time[k];
            pos = k;
        }
    }
    return pos;
}

void LRU(int pages[], int frames[], int time[], int fC, int pC) {
  printf("\nRef.String   |\tFrames\n");
  printf("-------------------------------\n");
  int i, j, k, pos, flag, faultCount, counter, queue;
  counter = 0, queue = 0, faultCount = 0;
  for (i = 0; i < pC; ++i) {
    flag = 0;
    printf("  %d\t|\t", pages[i]);
    for (j = 0; j < fC; ++j) {
      if (frames[j] == pages[i]) {
        flag = 1;
        counter++;
        time[j] = counter;
        printf("   Hit\n\n");
        break;
      }
    }
    if ((flag == 0) && (queue < fC)) {
      faultCount++;
      counter++;
      frames[queue] = pages[i];
      time[queue] = counter;
      queue++;
    }

    else if ((flag == 0) && (queue == fC)) {
      faultCount++;
      counter++;
      pos = findLRU(time, fC);
      frames[pos] = pages[i];
      time[pos] = counter;
    }
    if (flag == 0) {
      for (k = 0; k < fC; ++k) {
        printf("%d  ", frames[k]);
      }
      printf("\n\n");
    }
  }
  printf("\n Total Page Faults = %d\n\n", faultCount);
}

int main() {
  int i, pC, fC, pages[30], frames[20], time[20];
  printf("\n LRU \n");
  printf("\n Number of Frames : ");
  scanf("%d", &fC);
  for (i = 0; i < fC; ++i)
    frames[i] = -1;
  printf("\n Number of Pages : ");
  scanf("%d", &pC);
  printf("\n Enter the reference string : ");
  for (i = 0; i < pC; ++i)
    scanf("%d", &pages[i]);
  LRU(pages, frames, time, fC, pC);
  return 0;
}
```



```
 LRU

 Number of Frames : 3

 Number of Pages : 5

 Enter the reference string : 1 2 3 1 5

Ref.String   |  Frames
-------------------------------
  1     |       1  -1  -1

  2     |       1  2  -1

  3     |       1  2  3

  1     |          Hit

  5     |       1  5  3


 Total Page Faults = 4
```

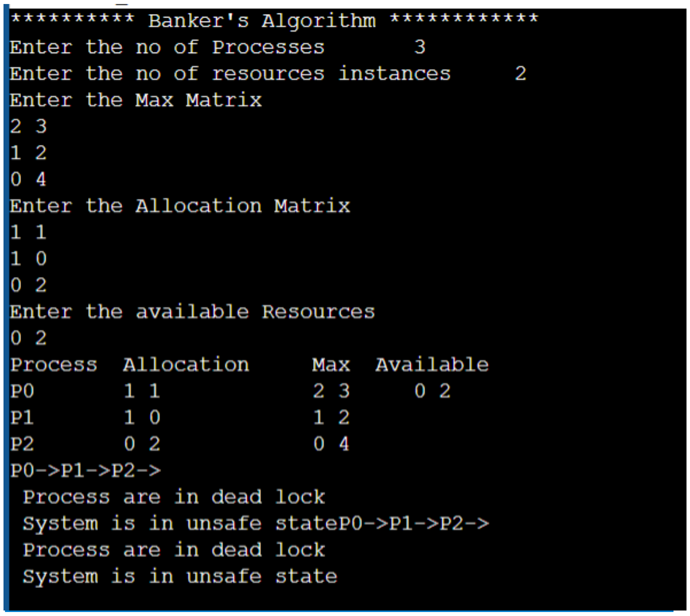
```
//LFU
#include<stdio.h>
int main()
{
int f,p;
int pages[50],frame[10],hit=0,count[50],time[50];
int i,j,page,flag,least,minTime,temp;
printf("Enter no of frames : ");
scanf("%d",&f);
printf("Enter no of pages : ");
scanf("%d",&p);
for(i=0;i<f;i++)
{
frame[i]=-1;
}
for(i=0;i<50;i++)
{
count[i]=0;
}
printf("Enter page no : \n");
for(i=0;i<p;i++)
{
   scanf("%d",&pages[i]);
}
printf("\n");
for(i=0;i<p;i++)
{
count[pages[i]]++;
time[pages[i]]=i;
flag=1;
least=frame[0];
for(j=0;j<f;j++)
{
if(frame[j]==-1 || frame[j]==pages[i])
{
if(frame[j]!=-1)
{
hit++;
}
flag=0;
frame[j]=pages[i];
break;
}
if(count[least]>count[frame[j]])
{
least=frame[j];
}
}
if(flag)
{
minTime=50;
for(j=0;j<f;j++)
{
if(count[frame[j]]==count[least] && time[frame[j]]<minTime)
{
temp=j;
minTime=time[frame[j]];
}
}
count[frame[temp]]=0;
frame[temp]=pages[i];
}
for(j=0;j<f;j++)
{
printf("%d ",frame[j]);
}
printf("\n");
}
printf("Page hit = %d",hit);
return 0;
}
```



```
Enter no of frames : 3
Enter no of pages : 5
Enter page no :
1
1
2
3
4

1 -1 -1
1 -1 -1
1 2 -1
1 2 3
1 4 3
○ Page hit = 1[redacted]
```

```
//Bankers algo
#include<stdio.h>
//#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
int i,j;
printf("********** Banker's Algorithm ************\n");
input();
show();
cal();
return 0;
}
void input()
{
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resources instances\t");
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]);
   }
```

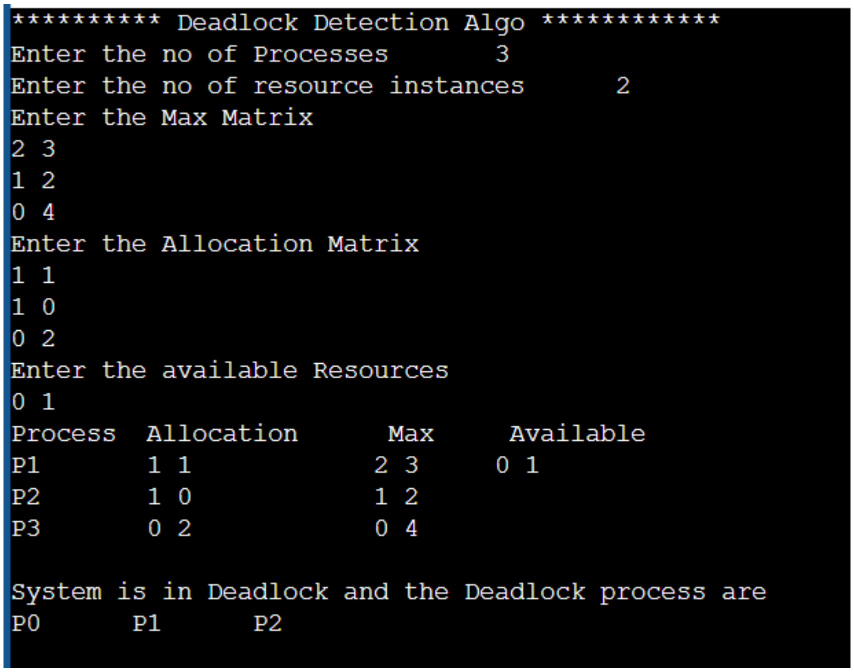
```
}
void show()
{
int i,j;
printf("Process\t Allocation  Max  Available\t");
for(i=0;i<n;i++)
   {
    printf("\nP%d\t\t ",i);
    for(j=0;j<r;j++)
       {
        printf("%d ",alloc[i][j]);
       }
    printf("\t");
    for(j=0;j<r;j++)
      {
       printf("%d ",max[i][j]);
      }
    printf("\t\t");
    if(i==0)
      {
       for(j=0;j<r;j++)
       printf("%d ",avail[j]);
      }
   }
}
void cal()
{
int finish[100],temp,need[100][100],flag=1,k,c1=0;
int safe[100];
int i,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][k];
                finish[i]=1;
                flag=1;
               }
             printf("P%d->",i);
            }
       }          }        }         }
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");
}}
```



```
********** Banker's Algorithm ************
Enter the no of Processes       3
Enter the no of resources instances     2
Enter the Max Matrix
2 3
1 2
0 4
Enter the Allocation Matrix
1 1
1 0
0 2
Enter the available Resources
0 2
Process  Allocation     Max  Available
P0       1 1            2 3     0 2
P1       1 0            1 2
P2       0 2            0 4
P0->P1->P2->
 Process are in dead lock
 System is in unsafe stateP0->P1->P2->
 Process are in dead lock
 System is in unsafe state
```

```
//Deadlock detection
#include<stdio.h>
#include<curses.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
int i,j;
printf("********** Deadlock Detection Algo ************\n");
input();
show();
cal();
getch();
return 0;
}
void input()
{int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resource instances\t");
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]);
}}
void show() {
int i,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]);
}}}
void cal()
{ int finish[100],temp,need[100][100],flag=1,k,c1=0;
int dead[100];
int safe[100];
int i,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)
```

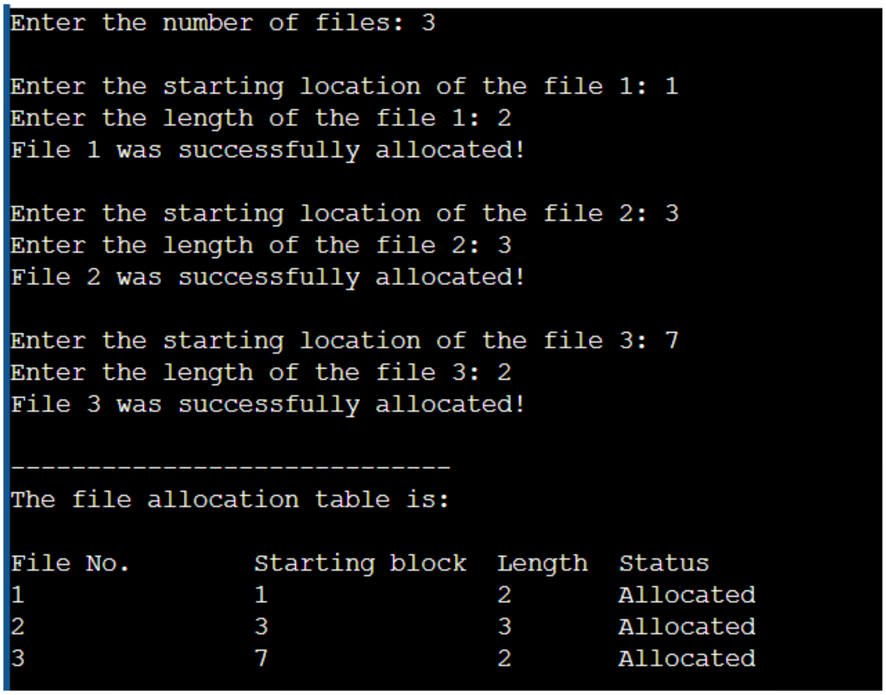
```
{
printf("\n\nSystem is in Deadlock and the Deadlock process are\n");
for(i=0;i<n;i++)
{printf("P%d\t",dead[i]);
}}
else
{
printf("\nNo Deadlock Occur"); }}
```



```
********** Deadlock Detection Algo ************
Enter the no of Processes       3
Enter the no of resource instances      2
Enter the Max Matrix
2 3
1 2
0 4
Enter the Allocation Matrix
1 1
1 0
0 2
Enter the available Resources
0 1
Process  Allocation      Max     Available
P1       1 1            2 3     0 1
P2       1 0            1 2
P3       0 2            0 4

System is in Deadlock and the Deadlock process are
P0      P1      P2
```

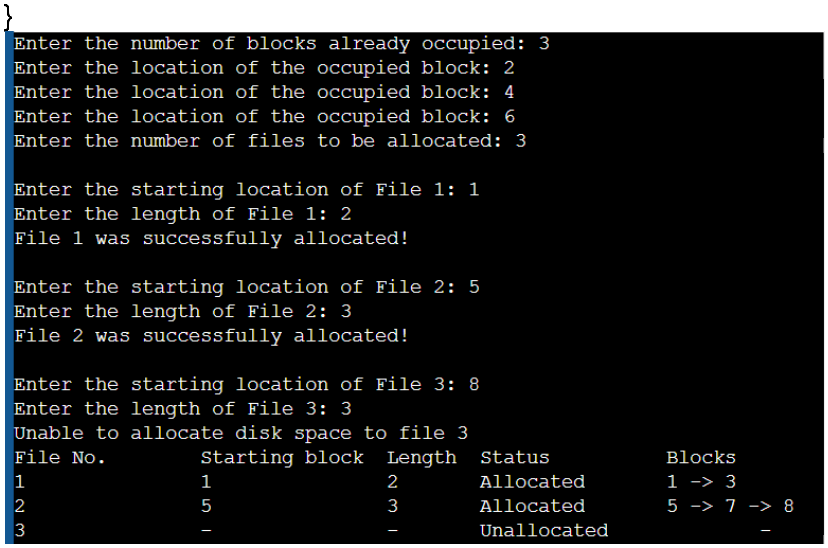
```
// File allocation contiguous
#include <stdio.h>

int files[100], start[10], len[10], alloc[10];

void allocate(int fno) {
    int count = 0;
    for(int i=start[fno]; i < (start[fno] + len[fno]); i++)
        if(files[i] == 0)
            count++;

    if(count == len[fno]) {
        for(int i=start[fno]; i < (start[fno] + len[fno]); i++)
            files[i] = 1;
        alloc[fno] = 1;
    }
    else
        alloc[fno] = 0;
}

void display(int n) {
    printf("\nFile No.\tStarting block\tLength\tStatus\n");
    for(int i=0; i < n; i++) {
        if(alloc[i] == 1)
            printf("%d\t\t%d\t\t%d\tAllocated\n", (i+1), start[i], len[i]);
        else
            printf("%d\t\t-\t\t-\tUnallocated\n", (i+1));
    }
}

int main() {
    int n;

    for(int i=0;i<100;i++)
        files[i] = 0;

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

    for(int i=0;i<n;i++) {
        printf("\nEnter the starting location of the file %d: ", (i+1));
        scanf("%d", &start[i]);
        printf("Enter the length of the file %d: ", (i+1));
        scanf("%d", &len[i]);
        allocate(i);
        if(alloc[i] == 1)
            printf("File %d was successfully allocated!\n", (i+1));
        else
```

```
                                        printf("Unable to allocate disk space to File %d\n", (i+1));
                }
                printf("\n-----------------------------\n");
                printf("The file allocation table is: \n");
                display(n);
        return 0;
}
```



```
Enter the number of files: 3

Enter the starting location of the file 1: 1
Enter the length of the file 1: 2
File 1 was successfully allocated!

Enter the starting location of the file 2: 3
Enter the length of the file 2: 3
File 2 was successfully allocated!

Enter the starting location of the file 3: 7
Enter the length of the file 3: 2
File 3 was successfully allocated!

-----------------------------
The file allocation table is:

File No.        Starting block  Length  Status
1               1               2       Allocated
2               3               3       Allocated
3               7               2       Allocated
```

```
//File allocation linked
#include <stdio.h>
#define MAX 25

int blocks[MAX];
typedef struct {
  int start;
  int len;
  int alloc[25];
  int flag;
}files;

files file[10];

void allocate(int fno) {
  int i = file[fno].start;
  int count = 0;

  do {
    if((i == file[fno].start) && (blocks[i] == 1)) {
      file[fno].flag = 0;
      break;
    }

    if(blocks[i] == 0) {
      blocks[i] = 1;
      file[fno].alloc[count] = i;
      count++;
    }
    i = (i+1) % MAX;
  }while(i!=file[fno].start && count<file[fno].len);

  if(count == file[fno].len)
    file[fno].flag = 1;
  else
    file[fno].flag = 0;
}

void display(int n) {
  int i,j;
  printf("File No.\tStarting block\tLength\tStatus\t\tBlocks\n");
  for (i = 0; i < n; i++) {
    if(file[i].flag == 1) {
      printf("%d\t\t%d\t\t%d\tAllocated\t", (i+1), file[i].start, file[i].len);
      for(j=0; j < file[i].len-1; j++)
        printf("%d -> ", file[i].alloc[j]);
      printf("%d\n", file[i].alloc[j]);
    }
    else {
      printf("%d\t\t-\t\t-\tUnallocated\t\t-\n", (i+1));
    }
  }
}

int main() {
  int n, filled, x;
  for(int i=0;i<MAX;i++)
    blocks[i] = 0;

  printf("Enter the number of blocks already occupied: ");
  scanf("%d", &filled);
  for(int i=0; i<filled; i++) {
    printf("Enter the location of the occupied block: ");
    scanf("%d", &x);
    blocks[x] = 1;
  }

  printf("Enter the number of files to be allocated: ");
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    printf("\nEnter the starting location of File %d: ", (i+1));
    scanf("%d", &file[i].start);
    printf("Enter the length of File %d: ", (i+1));
    scanf("%d", &file[i].len);
    allocate(i);
    if(file[i].flag == 1)
      printf("File %d was successfully allocated!\n", (i+1));
    else
      printf("Unable to allocate disk space to file %d\n", (i+1));
  }
  display(n);
  return 0;
}
```

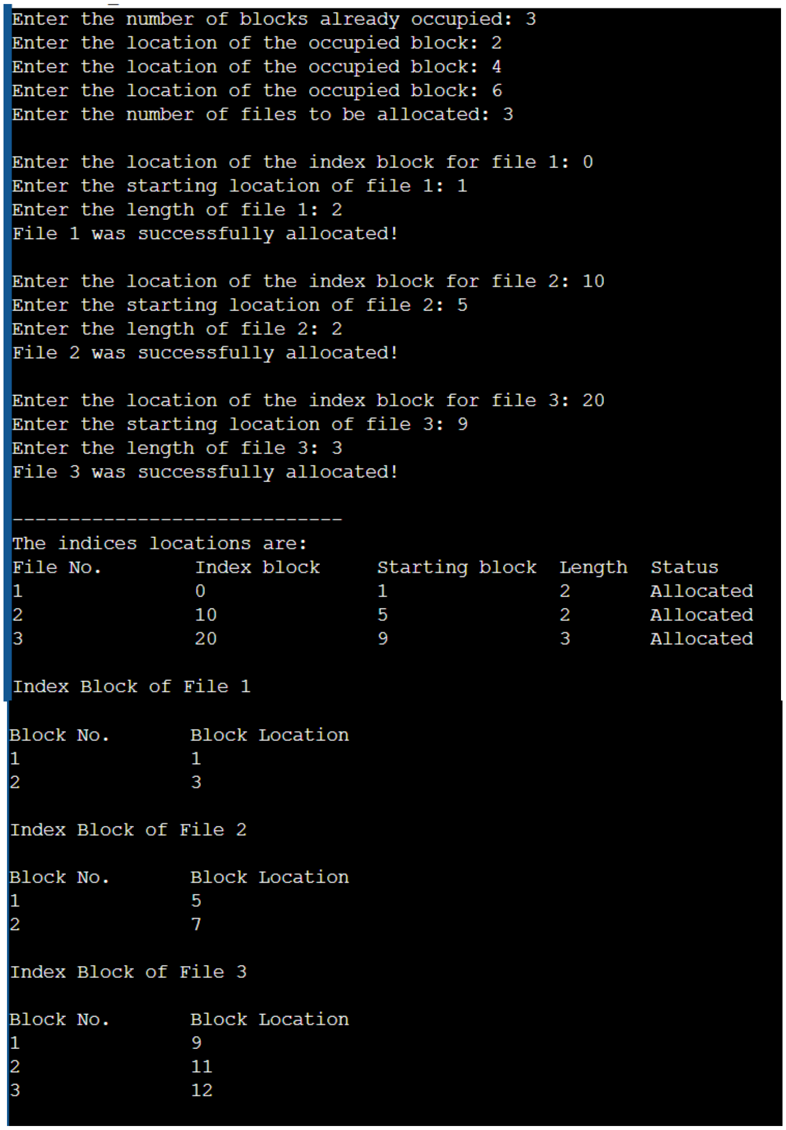


```
Enter the number of blocks already occupied: 3
Enter the location of the occupied block: 2
Enter the location of the occupied block: 4
Enter the location of the occupied block: 6
Enter the number of files to be allocated: 3

Enter the starting location of File 1: 1
Enter the length of File 1: 2
File 1 was successfully allocated!

Enter the starting location of File 2: 5
Enter the length of File 2: 3
File 2 was successfully allocated!

Enter the starting location of File 3: 8
Enter the length of File 3: 3
Unable to allocate disk space to file 3
File No.        Starting block  Length  Status          Blocks
1               1               2       Allocated       1 -> 3
2               5               3       Allocated       5 -> 7 -> 8
3               -               -       Unallocated             -
```

```
// Indexed File allocation
#include <stdio.h>
#define MAX 100

int blocks[MAX];
int indices[10];
typedef struct {
        int start;
        int len;
        int alloc[25];
        int flag;
}files;

files file[10];

void allocate(int fno) {
        int i = file[fno].start;
        int count = 0;

        do {
                if((i == file[fno].start) && (blocks[i] == 1)) {
                        file[fno].flag = 0;
                        break;
                }

                if(blocks[i] == 0) {
                        blocks[i] = 1;
                        file[fno].alloc[count] = i;
                        count++;
                }
                i = (i+1) % MAX;
        }while(i!=file[fno].start && count<file[fno].len);

        if(count == file[fno].len)
                file[fno].flag = 1;
        else
                file[fno].flag = 0;
}

void display(int n) {
        int i,j;
        printf("\n------------------------------\n");
        printf("The indices locations are: \n");
        printf("File No.\tIndex block\tStarting block\tLength\tStatus\n");
        for (int i = 0; i < n; i++) {
                if(indices[i] != -1 && file[i].flag == 1)
```

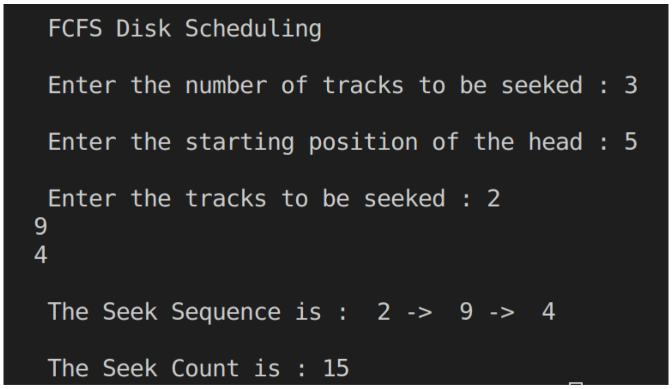
```
            printf("%d\t\t%d\t\t%d\t\t%d\tAllocated\n", (i+1), indices[i], file[i].start,
file[i].len);
        else
            printf("%d\t\t-\t\t-\t\t-\tUnallocated\n", (i+1));
    }
    for (i = 0; i < n; i++) {
        if(file[i].flag == 1)       {
            printf("\nIndex Block of File %d\n", (i+1));
            printf("\nBlock No.\tBlock Location\n");
            for(j=0; j < file[i].len; j++)
                printf("%d\t\t%d\n", (j+1), file[i].alloc[j]);
        }
    }
}

int main()
{
    int n, filled, x;
    for(int i=0;i<MAX;i++)
        blocks[i] = 0;

    printf("Enter the number of blocks already occupied: ");
    scanf("%d", &filled);
    for(int i=0; i<filled; i++) {
        printf("Enter the location of the occupied block: ");
        scanf("%d", &x);
        blocks[x] = 1;
    }

    printf("Enter the number of files to be allocated: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("\nEnter the location of the index block for file %d: ", (i+1));
        scanf("%d", &x);
        if(blocks[x] == 0) {
            blocks[x] = 1;
            indices[i] = x;
        }
        else {
            indices[i] = -1;
            printf("Index block is already occupied! Unable to store file %d\n", (i+1));
            continue;
        }
        printf("Enter the starting location of file %d: ", (i+1));
        scanf("%d", &file[i].start);
        printf("Enter the length of file %d: ", (i+1));
        scanf("%d", &file[i].len);
        allocate(i);
        if(file[i].flag == 1)
```

```
                                printf("File %d was successfully allocated!\n", (i+1));
                        else
                                printf("Starting location already occupied! Unable to allocate disk space to file %d\n", (i+1));
                }
                display(n);
  return 0;
}
```



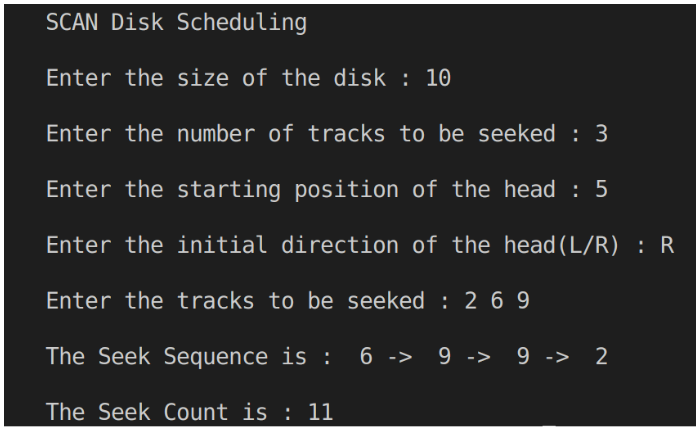
```
Enter the number of blocks already occupied: 3
Enter the location of the occupied block: 2
Enter the location of the occupied block: 4
Enter the location of the occupied block: 6
Enter the number of files to be allocated: 3

Enter the location of the index block for file 1: 0
Enter the starting location of file 1: 1
Enter the length of file 1: 2
File 1 was successfully allocated!

Enter the location of the index block for file 2: 10
Enter the starting location of file 2: 5
Enter the length of file 2: 2
File 2 was successfully allocated!

Enter the location of the index block for file 3: 20
Enter the starting location of file 3: 9
Enter the length of file 3: 3
File 3 was successfully allocated!

-----------------------------
The indices locations are:
File No.        Index block     Starting block  Length  Status
1               0               1               2       Allocated
2               10              5               2       Allocated
3               20              9               3       Allocated

Index Block of File 1

Block No.       Block Location
1               1
2               3

Index Block of File 2

Block No.       Block Location
1               5
2               7

Index Block of File 3

Block No.       Block Location
1               9
2               11
3               12
```

```
//FCFS disk scheduling
#include <stdio.h>
#include <stdlib.h>
#define MAX 25
int n, head, seek_count, tracks[MAX];
void fcfsds(){
  int curr_track, distance;
  seek_count = 0;
  for (int i = 0; i < n; i++){
    curr_track = tracks[i];
    distance = abs(head - curr_track);
    seek_count += distance;
    head = curr_track;
  }
}
int main() {
  printf("\n FCFS Disk Scheduling\n");
  printf("\n Enter the number of tracks to be seeked : ");
  scanf("%d", &n);
  if (n > MAX) {
    printf("\n Number of tracks to be seeked cannot exceed %d. Exiting...\n", MAX);
    exit(0);
  }
  printf("\n Enter the starting position of the head : ");
  scanf("%d", &head);
  printf("\n Enter the tracks to be seeked : ");
  for (int i = 0; i < n; i++)
    scanf("%d", &tracks[i]);
  fcfsds();
  printf("\n The Seek Sequence is : ");
  for (int i = 0; i < n - 1; i++)
    printf(" %d -> ", tracks[i]);

  printf(" %d\n", tracks[n - 1]);
  printf("\n The Seek Count is : %d\n", seek_count);
  return (0);
}
```



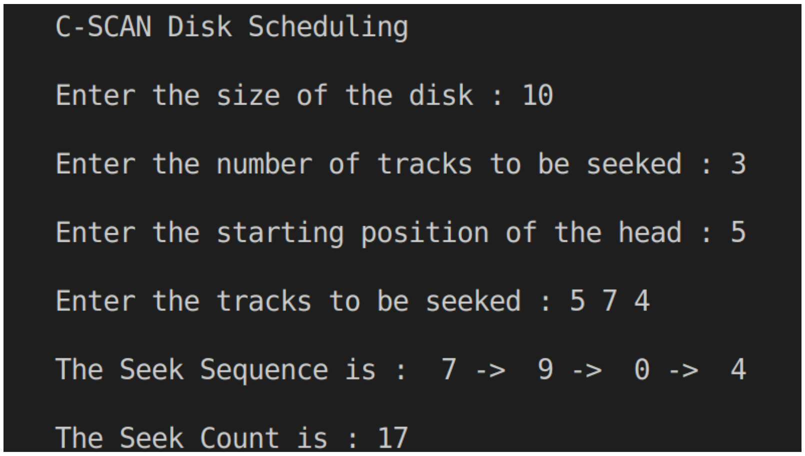
```
 FCFS Disk Scheduling

 Enter the number of tracks to be seeked : 3

 Enter the starting position of the head : 5

 Enter the tracks to be seeked : 2
9
4

 The Seek Sequence is :  2 ->  9 ->  4

 The Seek Count is : 15
```

```
//SCAN disk scheduling
#include <stdio.h>
#include <stdlib.h>

#define MAX 25

int n, head, size, seek_count, tracks[MAX], sequence[MAX];
char dir;

void sort(int arr[], int m){
  int temp;
  for (int i = 0; i < m; i++){
    for (int j = 0; j < m - 1 - i; j++){
      if (arr[j] > arr[j + 1]){
        temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}

void scands(){
  int curr_track, distance, l = 0, r = 0, left[MAX], right[MAX];
  seek_count = 0;
  if (dir == 'L'){
    left[0] = 0;
    l++;
  } else if (dir == 'R') {
    right[0] = size - 1;
    r++;
  }
  for (int i = 0; i < n; i++){
    if (tracks[i] < head)
      left[l++] = tracks[i];
    if (tracks[i] > head)
      right[r++] = tracks[i];
  }

  sort(left, l);
  sort(right, r);

  int run = 2, x = 0;

  while (run-- > 0){
    if (dir == 'L'){
      for (int i = l - 1; i >= 0; i--){
        curr_track = left[i];
        sequence[x++] = curr_track;
        distance = abs(head - curr_track);
        seek_count += distance;
        head = curr_track;
      }
      dir = 'R';
    } else {
        for (int i = 0; i < r; i++) {
          curr_track = right[i];
          sequence[x++] = curr_track;
          distance = abs(head - curr_track);
          seek_count += distance;
          head = curr_track;
        }
        dir = 'L';
      }
    }
}

int main(){
  int i;
  printf("\n SCAN Disk Scheduling\n");
  printf("\n Enter the size of the disk : ");
  scanf("%d", &size);
  printf("\n Enter the number of tracks to be seeked : ");
  scanf("%d", &n);
  if (n > MAX){
    printf("\n Number of tracks to be seeked cannot exceed %d Exiting...\n", MAX);
    exit(0);
  }
  printf("\n Enter the starting position of the head : ");
  scanf("%d", &head);
  if (head > size) {
    printf("\n Starting position of head cannot exceed the size of disk. Exiting...\n");
    exit(0);
  }
  printf("\n Enter the initial direction of the head(L/R) : ");
  scanf(" %c", &dir);
  if ((dir != 'L') && (dir != 'R')){
    printf("\n Invalid direction input. Exiting...\n");
    exit(0);
  }
  printf("\n Enter the tracks to be seeked : ");
  for (int i = 0; i < n; i++)
    scanf("%d", &tracks[i]);

  scands();
  printf("\n The Seek Sequence is : ");
  for (i = 0; i < n; i++)
    printf(" %d -> ", sequence[i]);
  printf(" %d\n", sequence[i]);
  printf("\n The Seek Count is : %d\n", seek_count);
  return 0;
}
```



```
SCAN Disk Scheduling

Enter the size of the disk : 10

Enter the number of tracks to be seeked : 3

Enter the starting position of the head : 5

Enter the initial direction of the head(L/R) : R

Enter the tracks to be seeked : 2 6 9

The Seek Sequence is :  6 ->  9 ->  9 ->  2

The Seek Count is : 11
```

```
//CSCAN disk scheduling
#include <stdio.h>
#include <stdlib.h>

#define MAX 25

int n, head, size, seek_count, tracks[MAX], sequence[MAX];

void sort(int arr[], int m){
  int temp;
  for (int i = 0; i < m; i++){
    for (int j = 0; j < m - 1 - i; j++){
      if (arr[j] > arr[j + 1]){
        temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }    }  }
}
void cscands(){
  int curr_track, distance, l, r, left[MAX], right[MAX];
  seek_count = 0;
  l = 0;
  r = 0;
  left[0] = 0;
  l++;
  right[0] = size - 1;
  r++;
  for (int i = 0; i < n; i++){
    if (tracks[i] < head)
      left[l++] = tracks[i];
    if (tracks[i] > head)
      right[r++] = tracks[i];
  }
  sort(left, l);
  sort(right, r);
  int x = 0;
  for (int i = 0; i < r; i++){
    curr_track = right[i];
    sequence[x++] = curr_track;
    distance = abs(head - curr_track);
    seek_count += distance;
    head = curr_track;
  }
  head = 0;
  seek_count += size - 1;
  for (int i = 0; i < l; i++){
    curr_track = left[i];
    sequence[x++] = curr_track;
    distance = abs(head - curr_track);
    seek_count += distance;
    head = curr_track;
  }
}

int main(){
  int i;
  printf("\n C-SCAN Disk Scheduling\n");
  printf("\n Enter the size of the disk : ");
  scanf("%d", &size);
  printf("\n Enter the number of tracks to be seeked : ");
  scanf("%d", &n);
  if (n > MAX){
    printf("\n Number of tracks to be seeked cannot exceed %d Exiting...\n", MAX);
    exit(0);
  }
  printf("\n Enter the starting position of the head : ");
  scanf("%d", &head);
  if (head > size) {
    printf("\n Starting position of head cannot exceed the size of disk. Exiting...\n");
    exit(0);
  }
  printf("\n Enter the tracks to be seeked : ");
  for (int i = 0; i < n; i++)
    scanf("%d", &tracks[i]);

  cscands();

  printf("\n The Seek Sequence is : ");
  for (i = 0; i < n; i++)
    printf(" %d -> ", sequence[i]);
  printf(" %d\n", sequence[i]);
  printf("\n The Seek Count is : %d\n", seek_count);
  return 0;
}
```



```
 C-SCAN Disk Scheduling

 Enter the size of the disk : 10

 Enter the number of tracks to be seeked : 3

 Enter the starting position of the head : 5

 Enter the tracks to be seeked : 5 7 4

 The Seek Sequence is :  7 ->  9 ->  0 ->  4

 The Seek Count is : 17
```