VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

OPERATING SYSTEMS

Submitted by

Snigdha S Kashyap 1BM21CS215

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
June-2023 to September-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "OPERATING SYSTEMS" carried out by **Snigdha S Kashyap (1BM21CS215),** who is a bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **OPERATING SYSTEMS (22CS4PCOPS)** work prescribed for the said degree.

Sonika Sharma D

Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak

Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course Outcome

CO1	Apply the different concepts and functionalities of Operating System.
CO2	Analyse various Operating system strategies and techniques.
CO3	Demonstrate the different functionalities of Operating System.
CO4	Conduct practical experiments to implement the functionalities of Operating system.

PROGRAM -1

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

- **FCFS**
- **SJF (pre-emptive & non-pre-emptive)**

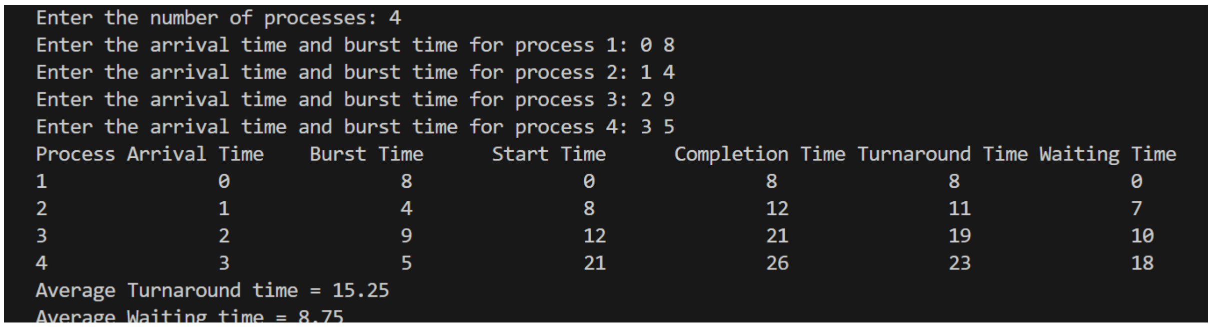
FCFS

```
#include <stdio.h>
typedef struct
{
   int pID, aT, bT, sT, cT, taT, wT;
} Process;

double avgTAT;
double avgWT;

void calculateTimes(Process p[], int n)
{

   int currT = 0;
   for (int i = 0; i < n; i++)
   {
      p[i].sT = currT;
      p[i].cT = currT + p[i].bT;
      p[i].taT = p[i].cT - p[i].aT;
      p[i].wT = p[i].taT - p[i].bT;
      currT = p[i].cT;
```

```
    }

    // To calculate Avg Turn Around Time and Avg Wating Time
    int sumTAT = 0;
    int sumWT = 0;

    for (int i = 0; i < n; i++)
    {
        sumTAT += p[i].taT;
        sumWT += p[i].wT;
    }
    avgTAT = (double)sumTAT / n;
    avgWT = (double)sumWT / n;
}

void displayp(Process p[], int n)
{
    printf("Process\tArrival Time\tBurst Time\tStart Time\tCompletion Time\tTurnaround 
Time\tWaiting Time\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pID, p[i].aT,
               p[i].bT, p[i].sT, p[i].cT,
               p[i].taT, p[i].wT);
    }
    printf("Average Turnaround time = %.2f\n", avgTAT);
    printf("Average Waiting time = %.2f\n", avgWT);
}
int main()
```

```
{
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    Process p[n];
    for (int i = 0; i < n; i++)
    {
        printf("Enter the arrival time and burst time for process %d: ", i + 1);
        scanf("%d %d", &p[i].aT, &p[i].bT);
        p[i].pID = i + 1;
    }
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (p[j].aT > p[j + 1].aT)
            {
                Process temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
    calculateTimes(p, n);
    displayp(p, n);
    return 0;
}
```

OUTPUT:



```
Enter the number of processes: 4
Enter the arrival time and burst time for process 1: 0 8
Enter the arrival time and burst time for process 2: 1 4
Enter the arrival time and burst time for process 3: 2 9
Enter the arrival time and burst time for process 4: 3 5
Process Arrival Time    Burst Time      Start Time      Completion Time Turnaround Time Waiting Time
1               0               8               0               8               8               0
2               1               4               8               12              11              7
3               2               9               12              21              19              10
4               3               5               21              26              23              18
Average Turnaround time = 15.25
Average Waiting time = 8.75
```

SJF (pre-emptive & non-pre-emptive)

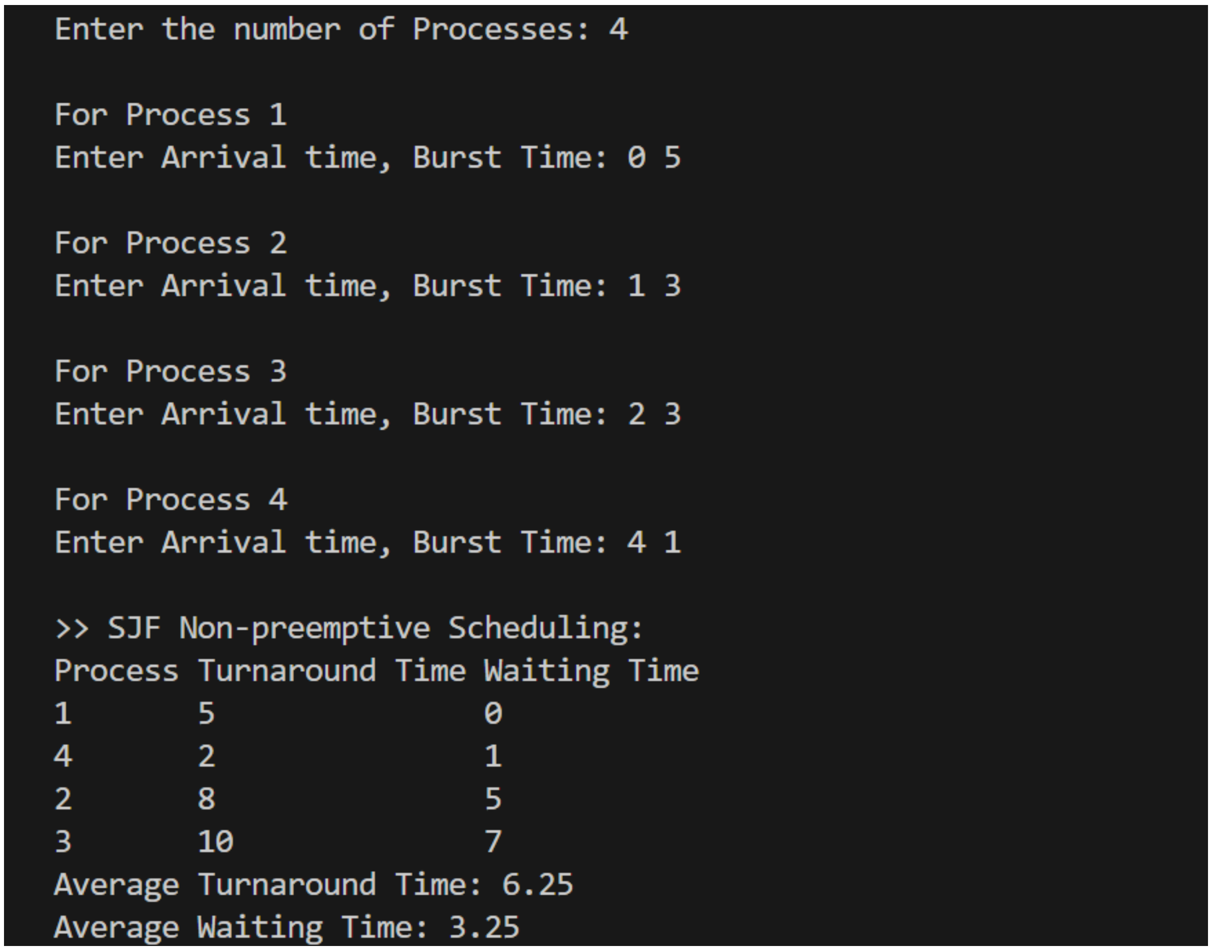
```
#include <stdio.h>
#include <stdbool.h>

#define MAX_PROCESSES 10

struct Process
{
    int pid;
    int arr_time;
    int burst_time;
    int rem_time;
    int tat;
    int wt;
};

void sjf_nonpreemptive(struct Process p[], int n)
{
    int i, j, count = 0, m;
    for (i = 0; i < n; i++)
    {
        if (p[i].arr_time == 0)
            count++;
    }
    if (count == n || count == 1)
    {
        if (count == n)
```

```
{
    for (i = 0; i < n - 1; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (p[j].burst_time > p[j + 1].burst_time)
            {
                struct Process temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}
else
{
    for (i = 1; i < n - 1; i++)
    {
        for (j = 1; j <= n - i - 1; j++)
        {
            if (p[j].burst_time > p[j + 1].burst_time)
            {
                struct Process temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
```

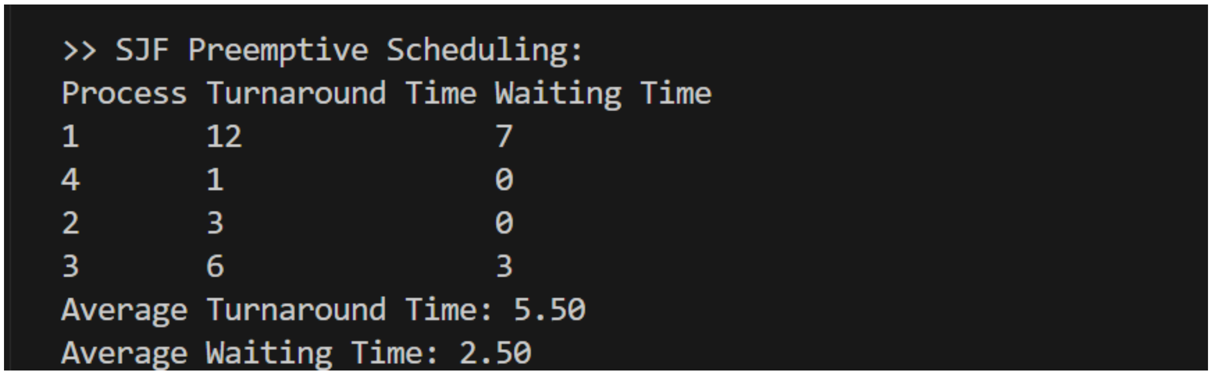
```
        }
    }

    int total_time = 0;
    double total_tat = 0;
    double total_wt = 0;

    for (i = 0; i < n; i++)
    {
        total_time += p[i].burst_time;
        p[i].tat = total_time - p[i].arr_time;
        p[i].wt = p[i].tat - p[i].burst_time;

        total_tat += p[i].tat;
        total_wt += p[i].wt;
    }

    printf("Process\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t%d\n", p[i].pid, p[i].tat, p[i].wt);
    }

    printf("Average Turnaround Time: %.2f\n", total_tat / n);
    printf("Average Waiting Time: %.2f\n", total_wt / n);
}

void sjf_preemptive(struct Process p[], int n)
```

```
{
    int total_time = 0, i;
    int completed = 0;

    while (completed < n)
    {
        int shortest_burst = -1;
        int next_process = -1;

        for (i = 0; i < n; i++)
        {
            if (p[i].arr_time <= total_time && p[i].rem_time > 0)
            {
                if (shortest_burst == -1 || p[i].rem_time < shortest_burst)
                {
                    shortest_burst = p[i].rem_time;
                    next_process = i;
                }
            }
        }

        if (next_process == -1)
        {
            total_time++;
            continue;
        }

        p[next_process].rem_time--;
        total_time++;

        if (p[next_process].rem_time == 0)
        {
            completed++;
            p[next_process].tat = total_time - p[next_process].arr_time;
            p[next_process].wt = p[next_process].tat - p[next_process].burst_time;
        }
    }

    double total_tat = 0;
    double total_wt = 0;

    printf("Process\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t%d\n", p[i].pid, p[i].tat, p[i].wt);
        total_tat += p[i].tat;
        total_wt += p[i].wt;
    }
    printf("Average Turnaround Time: %.2f\n", total_tat / n);
    printf("Average Waiting Time: %.2f\n", total_wt / n);
}

int main()
{
    int n, quantum, i, choice;
    struct Process p[MAX_PROCESSES];
    printf("Enter the number of Processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nFor Process %d\n", i + 1);
        printf("Enter Arrival time, Burst Time: ");
        scanf("%d%d", &p[i].arr_time, &p[i].burst_time);
        p[i].pid = i + 1;
        p[i].rem_time = p[i].burst_time;
        p[i].tat = 0;
        p[i].wt = 0;
    }
    printf("\n>> SJF Non-preemptive Scheduling:\n");
    sjf_nonpreemptive(p, n);
    printf("\n>> SJF Preemptive Scheduling:\n");
    sjf_preemptive(p, n);

    return 0;
}
```

OUTPUT:



```
Enter the number of Processes: 4

For Process 1
Enter Arrival time, Burst Time: 0 5

For Process 2
Enter Arrival time, Burst Time: 1 3

For Process 3
Enter Arrival time, Burst Time: 2 3

For Process 4
Enter Arrival time, Burst Time: 4 1

>> SJF Non-preemptive Scheduling:
Process Turnaround Time Waiting Time
1       5               0
4       2               1
2       8               5
3       10              7
Average Turnaround Time: 6.25
Average Waiting Time: 3.25
```



```
>> SJF Preemptive Scheduling:
Process Turnaround Time Waiting Time
1       12              7
4       1               0
2       3               0
3       6               3
Average Turnaround Time: 5.50
Average Waiting Time: 2.50
```

PROGRAM-2

Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.

- **Priority (pre-emptive & non-pre-emptive)**
- **Round Robin (Experiment with different quantum sizes for RR algorithm)**

```
#include <stdio.h>

#include <stdbool.h>


#define MAX_PROCESSES 10


struct Process

{

    int pid;

    int arr_time;

    int burst_time;

    int priority;

    int rem_time;

    int tat;

    int wt;

};



void priority_nonpreemptive(struct Process p[], int n)

{

    int i, j, count = 0, m;

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

    {

        if (p[i].arr_time == 0)
        count++;
    }
    if (count == n || count == 1)
    {
        if (count == n)
        {
            for (i = 0; i < n - 1; i++)
            {
                for (j = 0; j < n - i - 1; j++)
                {
                    if (p[j].priority > p[j + 1].priority)
                    {
                        struct Process temp = p[j];
                        p[j] = p[j + 1];
                        p[j + 1] = temp;
                    }
                }
            }
        }

        else
        {
            for (i = 1; i < n - 1; i++)
            {
                for (j = 1; j <= n - i - 1; j++)
                {
                    if (p[j].priority > p[j + 1].priority)
                    {
```

```
                struct Process temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

int total_time = 0;
double total_tat = 0;
double total_wt = 0;

for (i = 0; i < n; i++)
{
    total_time += p[i].burst_time;
    p[i].tat = total_time - p[i].arr_time;
    p[i].wt = p[i].tat - p[i].burst_time;

    total_tat += p[i].tat;
    total_wt += p[i].wt;
}

printf("Process\tTurnaround Time\tWaiting Time\n");
for (i = 0; i < n; i++)
{
    printf("%d\t%d\t\t%d\n", p[i].pid, p[i].tat, p[i].wt);
}
```

```
    printf("Average Turnaround Time: %.2f\n", total_tat / n);
    printf("Average Waiting Time: %.2f\n", total_wt / n);
}

void priority_preemptive(struct Process p[], int n)
{
    int total_time = 0, i;
    int completed = 0;

    while (completed < n)
    {
        int highest_priority = -1;
        int next_process = -1;

        for (i = 0; i < n; i++)
        {
            if (p[i].arr_time <= total_time && p[i].rem_time > 0)
            {
                if (highest_priority == -1 || p[i].priority < highest_priority)
                {
                    highest_priority = p[i].priority;
                    next_process = i;
                }
            }
        }

        if (next_process == -1)
    {
        total_time++;
        continue;
    }

    p[next_process].rem_time--;
    total_time++;

    if (p[next_process].rem_time == 0)
    {
        completed++;
        p[next_process].tat = total_time - p[next_process].arr_time;
        p[next_process].wt = p[next_process].tat - p[next_process].burst_time;
    }
}

double total_tat = 0;
double total_wt = 0;

printf("Process\tTurnaround Time\tWaiting Time\n");
for (i = 0; i < n; i++)
{
    printf("%d\t%d\t\t%d\n", p[i].pid, p[i].tat, p[i].wt);

    total_tat += p[i].tat;
    total_wt += p[i].wt;
}
```

```
    printf("Average Turnaround Time: %.2f\n", total_tat / n);
    printf("Average Waiting Time: %.2f\n", total_wt / n);
}

void round_robin(struct Process p[], int n, int quantum)
{
    int total_time = 0, i;
    int completed = 0;

    printf("\nGantt Chart: \n");
    while (completed < n)
    {

        for (i = 0; i < n; i++)
        {
            if (p[i].arr_time <= total_time && p[i].rem_time > 0)
            {
                if (p[i].rem_time <= quantum)
                {
                    printf("P%d ", p[i].pid);
                    total_time += p[i].rem_time;
                    p[i].rem_time = 0;
                    p[i].tat = total_time - p[i].arr_time;
                    p[i].wt = p[i].tat - p[i].burst_time;
                    completed++;
                }
                else
                {
```

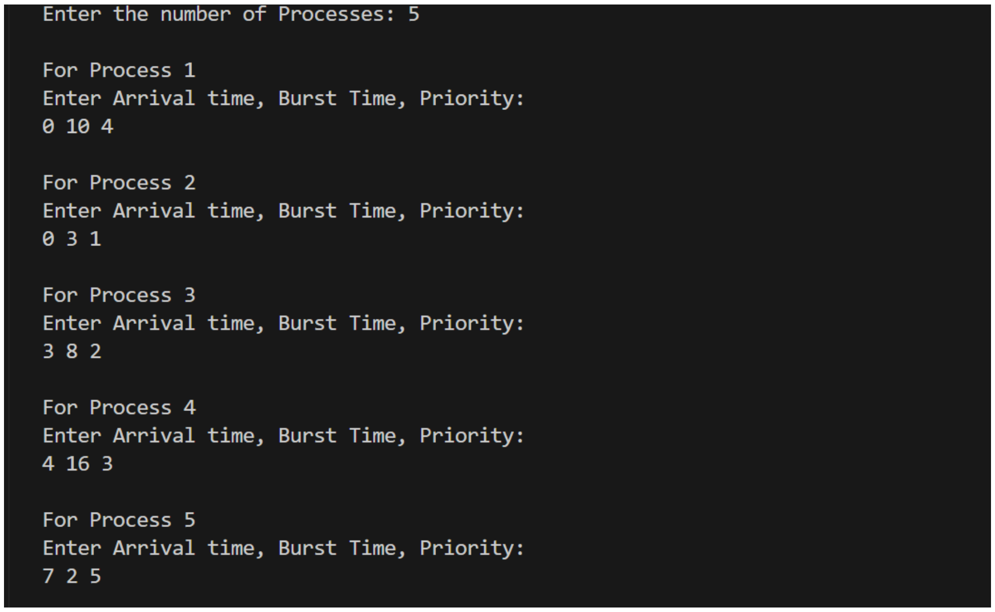
```
                printf("P%d ", p[i].pid);
                total_time += quantum;
                p[i].rem_time -= quantum;
            }
        }
    }
}

double total_tat = 0;
double total_wt = 0;

printf("\n");
printf("\nProcess\tTurnaround Time\tWaiting Time\n");
for (i = 0; i < n; i++)
{
    printf("%d\t%d\t\t%d\n", p[i].pid, p[i].tat, p[i].wt);

    total_tat += p[i].tat;
    total_wt += p[i].wt;
}

printf("Average Turnaround Time: %.2f\n", total_tat / n);
printf("Average Waiting Time: %.2f\n", total_wt / n);
}

int main()
{
    int n, quantum, i, choice;
```

```
struct Process p[MAX_PROCESSES];

printf("Enter the number of Processes: ");
scanf("%d", &n);
for (i = 0; i < n; i++)
{
   printf("\nFor Process %d\n", i + 1);
   printf("Enter Arrival time, Burst Time, Priority:\n");
   scanf("%d%d%d",&p[i].arr_time,&p[i].burst_time,&p[i].priority);
   p[i].pid = i + 1;
   p[i].rem_time = p[i].burst_time;
   p[i].tat = 0;
   p[i].wt = 0;
}

printf("\nSelect a scheduling algorithm:\n");

printf("1. Priority (Preemtive & Non-preemptive)\n");
printf("2. Round Robin\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice)
{
case 1:
   printf("\n>> Priority Non-preemptive Scheduling:\n");
   priority_nonpreemptive(p, n);
   printf("\n>> Priority Preemptive Scheduling:\n");
   priority_preemptive(p, n);
```

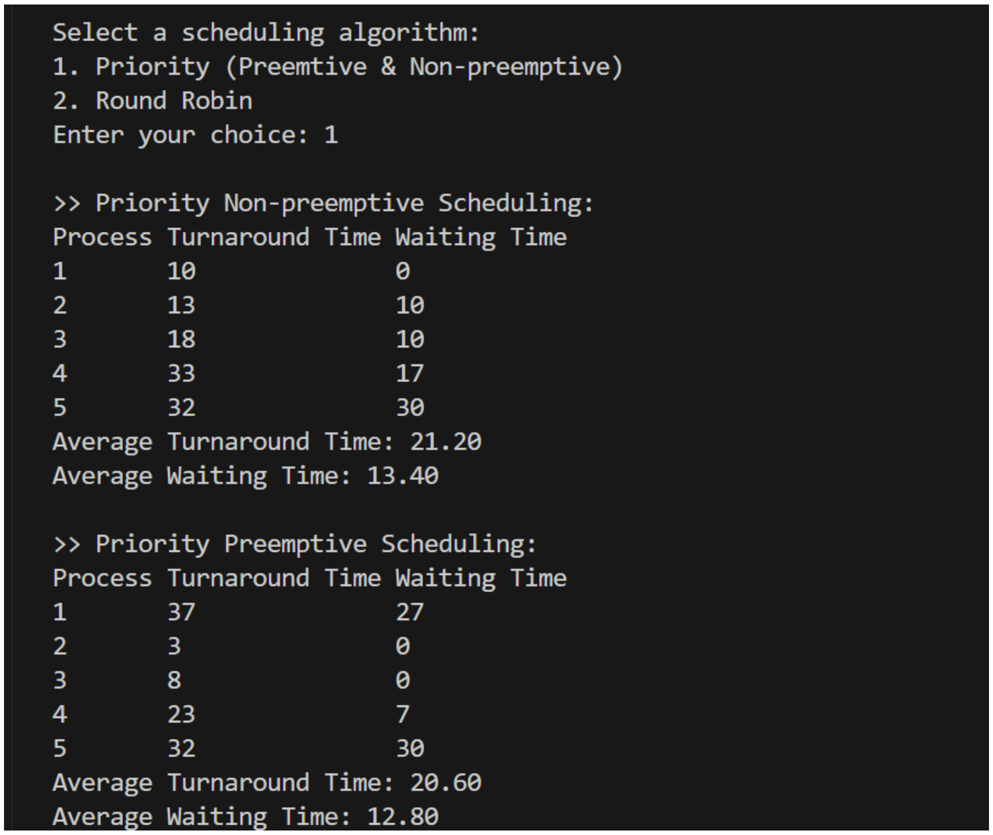
```
        break;
    case 2:
        printf("\nEnter the quantum size for Round Robin: ");
        scanf("%d", &quantum);
        printf("\n>> Round Robin Scheduling (Quantum: %d):\n", quantum);
        round_robin(p, n, quantum);
        break;
    default:
        printf("Invalid choice!\n");
        return 1;
    }

    return 0;
}
```

OUTPUT:

Priority (pre-emptive & non-pre-emptive):

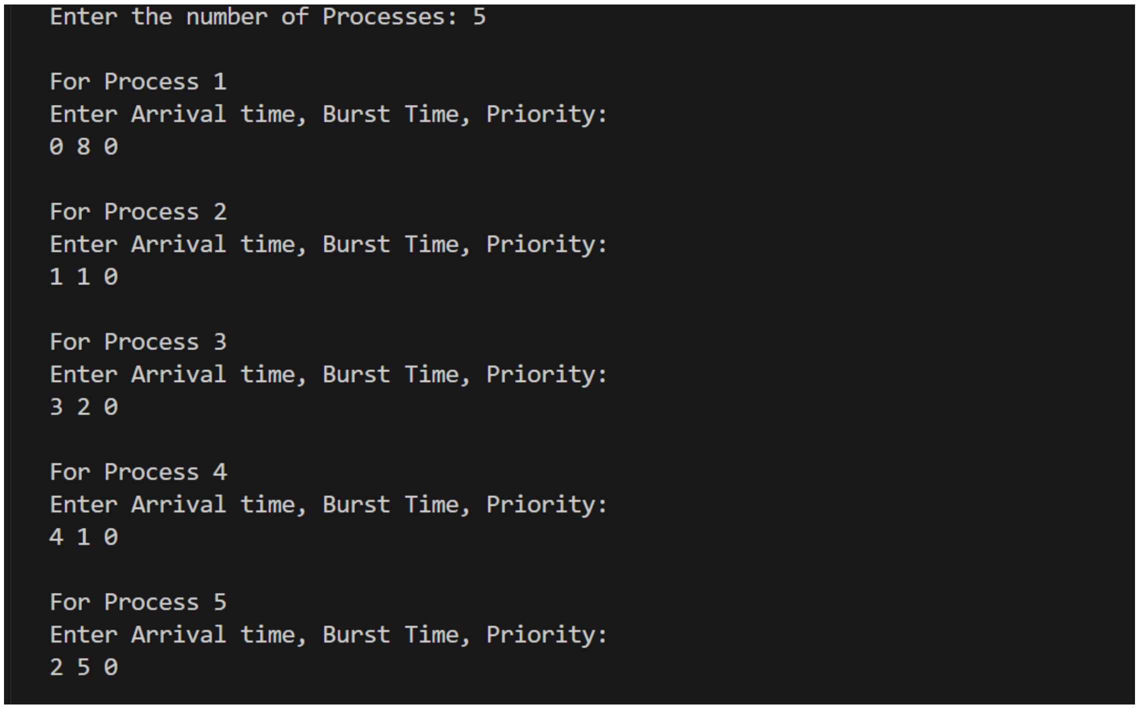


```
Enter the number of Processes: 5

For Process 1
Enter Arrival time, Burst Time, Priority:
0 10 4

For Process 2
Enter Arrival time, Burst Time, Priority:
0 3 1

For Process 3
Enter Arrival time, Burst Time, Priority:
3 8 2

For Process 4
Enter Arrival time, Burst Time, Priority:
4 16 3

For Process 5
Enter Arrival time, Burst Time, Priority:
7 2 5
```

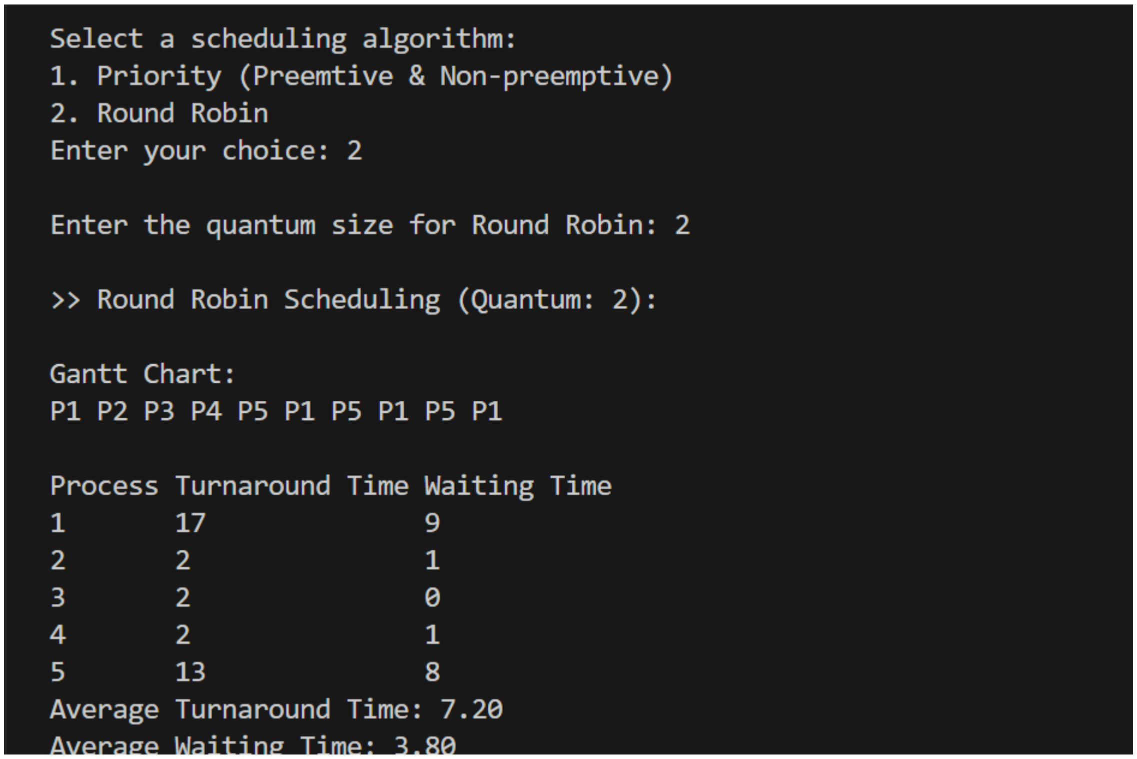


```
Select a scheduling algorithm:
1. Priority (Preemtive & Non-preemptive)
2. Round Robin
Enter your choice: 1

>> Priority Non-preemptive Scheduling:
Process Turnaround Time Waiting Time
1       10              0
2       13              10
3       18              10
4       33              17
5       32              30
Average Turnaround Time: 21.20
Average Waiting Time: 13.40

>> Priority Preemptive Scheduling:
Process Turnaround Time Waiting Time
1       37              27
2       3               0
3       8               0
4       23              7
5       32              30
Average Turnaround Time: 20.60
Average Waiting Time: 12.80
```

Round Robin:



```
Enter the number of Processes: 5

For Process 1
Enter Arrival time, Burst Time, Priority:
0 8 0

For Process 2
Enter Arrival time, Burst Time, Priority:
1 1 0

For Process 3
Enter Arrival time, Burst Time, Priority:
3 2 0

For Process 4
Enter Arrival time, Burst Time, Priority:
4 1 0

For Process 5
Enter Arrival time, Burst Time, Priority:
2 5 0
```



```
Select a scheduling algorithm:
1. Priority (Preemtive & Non-preemptive)
2. Round Robin
Enter your choice: 2

Enter the quantum size for Round Robin: 2

>> Round Robin Scheduling (Quantum: 2):

Gantt Chart:
P1 P2 P3 P4 P5 P1 P5 P1 P5 P1

Process Turnaround Time Waiting Time
1       17              9
2       2               1
3       2               0
4       2               1
5       13              8
Average Turnaround Time: 7.20
Average Waiting Time: 3.80
```

PROGRAM-3

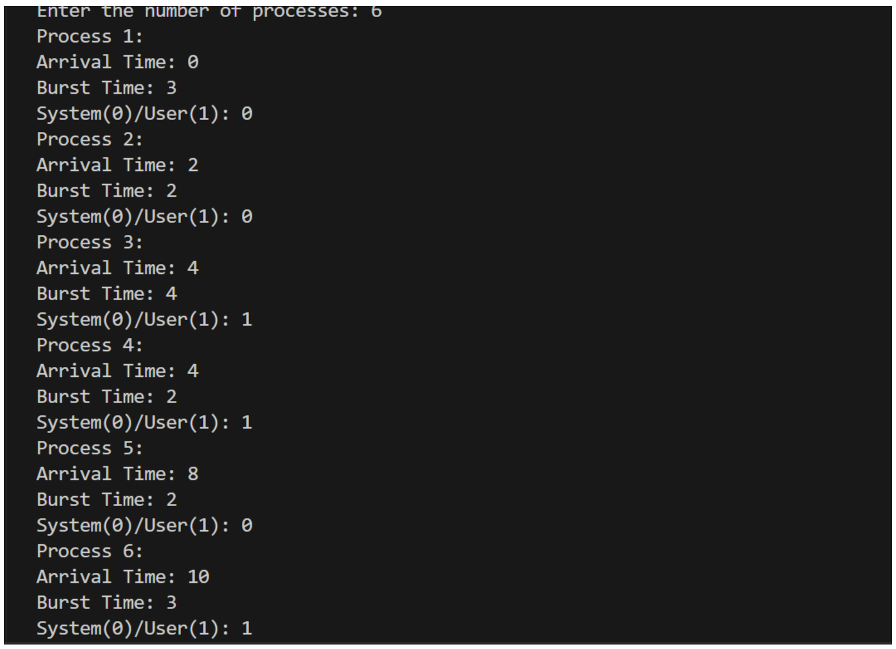
Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_QUEUE_SIZE 100


int totalTime = 0;

int userProcess = 0, systemProcess = 0;


typedef struct {

    int processID;

    int arrivalTime;

    int burstTime;

    int remainingTime;

    int priority; // 0 for system process, 1 for user process

} Process;


void executeProcess(Process process) {

    printf("Executing Process %d\n", process.processID);

    for (int i = 1; i <= process.burstTime; i++) {

        printf("Process %d: %d/%d\n", process.processID, i, process.burstTime);

    }

    printf("Process %d executed\n", process.processID);

}
```

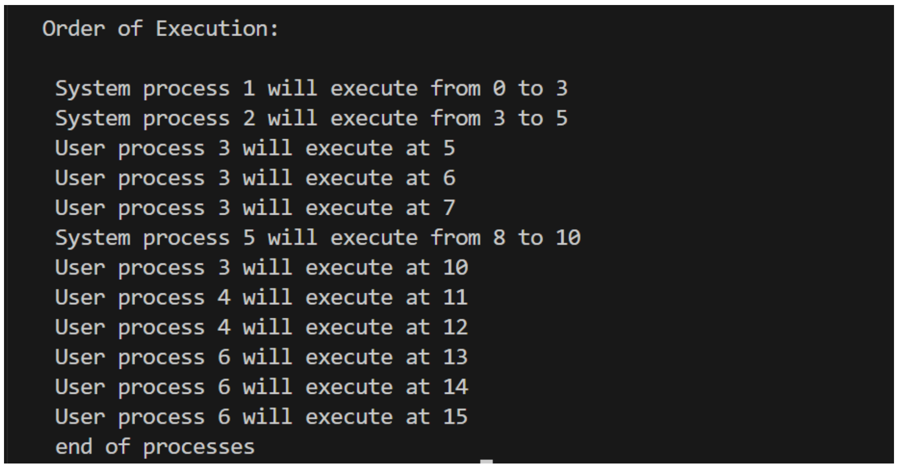
```
void scheduleFCFS(Process system[], Process user[]) {
    for (int i = 0; i < systemProcess; i++) {
        for (int j = i + 1; j < systemProcess; j++) {
            if (system[i].arrivalTime > system[j].arrivalTime) {
                Process temp = system[i];
                system[i] = system[j];
                system[j] = temp;
            }
        }
    }
    for (int i = 0; i < userProcess; i++) {
        for (int j = i + 1; j < userProcess; j++) {
            if (user[i].arrivalTime > user[j].arrivalTime) {
                Process temp = user[i];
                user[i] = user[j];
                user[j] = temp;
            }
        }
    }
    int completed = 0;
    int currentProcess = -1;
    int isUserProcess = 0; // Changed bool to int
    int size = userProcess + systemProcess;
    while (1) {
        int count = 0;
        for (int i = 0; i < systemProcess; i++) {
            if (system[i].remainingTime <= 0) {
                count++;
            }
        }
        for (int j = 0; j < userProcess; j++) {
            if (user[j].remainingTime <= 0) {
                count++;
            }
        }
        if (count == size) {
            printf("\n end of processes");
            exit(0);
        }
        for (int i = 0; i < systemProcess; i++) {
            if (totalTime >= system[i].arrivalTime && system[i].remainingTime > 0) {
                currentProcess = i;
                isUserProcess = 0; // Changed true to 0
                break;
            }
        }
        if (currentProcess == -1) {
            for (int j = 0; j < userProcess; j++) {
                if (totalTime >= user[j].arrivalTime && user[j].remainingTime > 0) {
                    currentProcess = j;
                    isUserProcess = 1; // Changed true to 1
                    break;
                }
            }
        }
        if (currentProcess == -1) {
```

```
            totalTime++;
            printf("\n %d   idle time...", totalTime);
            if (totalTime == 1000) {
                exit(0);
            }
            continue;
        }
        if (isUserProcess == 1) { // Changed true to 1
            user[currentProcess].remainingTime--;
            printf("\n User process %d will execute at %d ", user[currentProcess].processID,
(totalTime));
            totalTime++;
            isUserProcess = 0; // Changed true to 0
            currentProcess = -1;
            if (user[currentProcess].remainingTime == 0) {
                completed++;
            }
        } else {
            int temp = totalTime;
            while (system[currentProcess].remainingTime--) {
                totalTime++;
            }
            if (system[currentProcess].remainingTime == 0) {
                completed++;
            }
            printf("\n System process %d will execute from %d to %d ",
system[currentProcess].processID, temp, (totalTime));
            isUserProcess = 0; // Changed true to 0
            currentProcess = -1;
        }
    }
}

int main() {
    int numProcesses;
    Process processes[MAX_QUEUE_SIZE];

    // Reading the number of processes
    printf("Enter the number of processes: ");
    scanf("%d", &numProcesses);
    // Reading process details
    for (int i = 0; i < numProcesses; i++) {
        printf("Process %d:\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &processes[i].arrivalTime);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burstTime);
        printf("System(0)/User(1): ");
        scanf("%d", &processes[i].priority);
        processes[i].processID = i + 1;
        processes[i].remainingTime = processes[i].burstTime;
        if (processes[i].priority == 1) {
            userProcess++;
        } else {
            systemProcess++;
        }
    }
    Process systemQueue[MAX_QUEUE_SIZE];
    int systemQueueSize = 0;
    Process userQueue[MAX_QUEUE_SIZE];
    int userQueueSize = 0;
    for (int i = 0; i < numProcesses; i++) {
        if (processes[i].priority == 0) {
            systemQueue[systemQueueSize++] = processes[i];
        } else {
            userQueue[userQueueSize++] = processes[i];
        }
    }
    printf("Order of Execution:\n");
    scheduleFCFS(systemQueue, userQueue);
    return 0;
}
```

OUTPUT:



```
Enter the number of processes: 6
Process 1:
Arrival Time: 0
Burst Time: 3
System(0)/User(1): 0
Process 2:
Arrival Time: 2
Burst Time: 2
System(0)/User(1): 0
Process 3:
Arrival Time: 4
Burst Time: 4
System(0)/User(1): 1
Process 4:
Arrival Time: 4
Burst Time: 2
System(0)/User(1): 1
Process 5:
Arrival Time: 8
Burst Time: 2
System(0)/User(1): 0
Process 6:
Arrival Time: 10
Burst Time: 3
System(0)/User(1): 1
```



```
Order of Execution:

 System process 1 will execute from 0 to 3
 System process 2 will execute from 3 to 5
 User process 3 will execute at 5
 User process 3 will execute at 6
 User process 3 will execute at 7
 System process 5 will execute from 8 to 10
 User process 3 will execute at 10
 User process 4 will execute at 11
 User process 4 will execute at 12
 User process 6 will execute at 13
 User process 6 will execute at 14
 User process 6 will execute at 15
 end of processes
```

PROGRAM-4

Write a C program to simulate Real-Time CPU Scheduling algorithms:

a) Rate- Monotonic

b) Earliest-deadline First

a) Rate- Monotonic

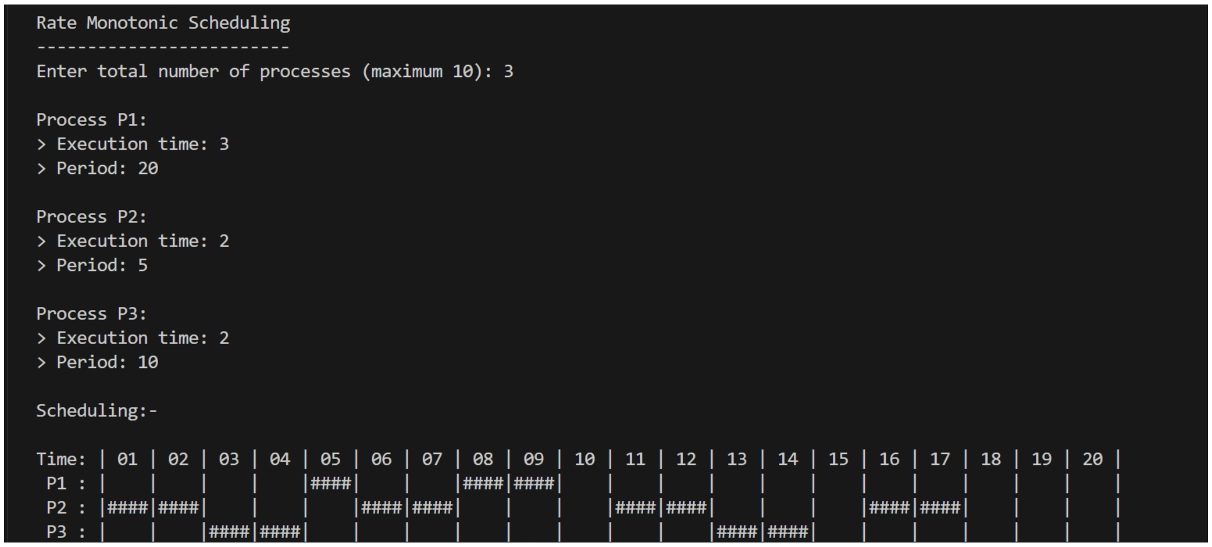
```
#include <stdio.h>

#include <math.h>

#include <stdlib.h>


#define MAX_PROCESS 10


int num_of_process = 3;

int execution_time[MAX_PROCESS], period[MAX_PROCESS],
remain_time[MAX_PROCESS];


// collecting details of processes

void get_process_info()

{

    printf("Enter total number of processes (maximum %d): ", MAX_PROCESS);

    scanf("%d", &num_of_process);

    if (num_of_process < 1)

    {

        printf("Do you really want to schedule %d processes? -_-\n", num_of_process);

        exit(0);

    }

    for (int i = 0; i < num_of_process; i++)

    {

        printf("\nProcess %d:-\n", i + 1);
        printf("==> Execution time: ");
        scanf("%d", &execution_time[i]);
        remain_time[i] = execution_time[i];

        printf("==> Period: ");
        scanf("%d", &period[i]);
    }
}

// get maximum of three numbers
int max(int a, int b, int c)
{
    if (a >= b && a >= c)
        return a;
    else if (b >= a && b >= c)
        return b;
    else
        return c;
}

// calculating the observation time for scheduling timeline
int get_observation_time()
{
    return max(period[0], period[1], period[2]);
}
// print scheduling sequence
void print_schedule(int process_list[], int cycles)
```

```
{
    printf("\nScheduling:-\n\n");
    printf("Time: ");
    for (int i = 0; i < cycles; i++)
    {
        if (i < 9)
            printf("| 0%d ", i + 1);
        else
            printf("| %d ", i + 1);
    }
    printf("|\n");

    for (int i = 0; i < num_of_process; i++)
    {
        printf("P[%d]: ", i + 1);
        for (int j = 0; j < cycles; j++)
        {
            if (process_list[j] == i + 1)
                printf("|####");
            else
                printf("|    ");
        }
        printf("|\n");
    }
}

void rate_monotonic(int time)
{
```

```
float utilization = 0;
for (int i = 0; i < num_of_process; i++)
{
    utilization += (1.0 * execution_time[i]) / period[i];
}
int n = num_of_process;
if (utilization > n * (pow(2, 1.0 / n) - 1))
{
    printf("\nGiven problem is not schedulable under said scheduling algorithm.\n");
    exit(0);
}

int process_list[time];
int min = 999, next_process = 0;
for (int i = 0; i < time; i++)
{
    min = 1000;
    for (int j = 0; j < num_of_process; j++)
    {
        if (remain_time[j] > 0)
        {
            if (min > period[j])
            {
                min = period[j];
                next_process = j;
            }
        }
    }
```

```
        if (remain_time[next_process] > 0)
        {
            process_list[i] = next_process + 1; // +1 for catering 0 array index.
            remain_time[next_process] -= 1;
        }

        for (int k = 0; k < num_of_process; k++)
        {
            if ((i + 1) % period[k] == 0)
            {
                remain_time[k] = execution_time[k];
                next_process = k;
            }
        }
    }
    print_schedule(process_list, time);
}

int main(int argc, char *argv[])
{
    printf("Rate Monotonic Scheduling\n");
    printf("--------------------------\n");

    get_process_info(); // collecting processes detail
    int observation_time = get_observation_time();
    rate_monotonic(observation_time);
    return 0;
}
```

<u>OUTPUT:</u>



```
Rate Monotonic Scheduling
-------------------------
Enter total number of processes (maximum 10): 3

Process P1:
> Execution time: 3
> Period: 20

Process P2:
> Execution time: 2
> Period: 5

Process P3:
> Execution time: 2
> Period: 10

Scheduling:-

Time: | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
 P1 : |    |    |    |    |####|    |    |####|####|    |    |    |    |    |    |    |    |    |    |    |
 P2 : |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |
 P3 : |    |    |####|####|    |    |    |    |    |    |    |    |####|####|    |    |    |    |    |    |
```

b) Earliest-Deadline First

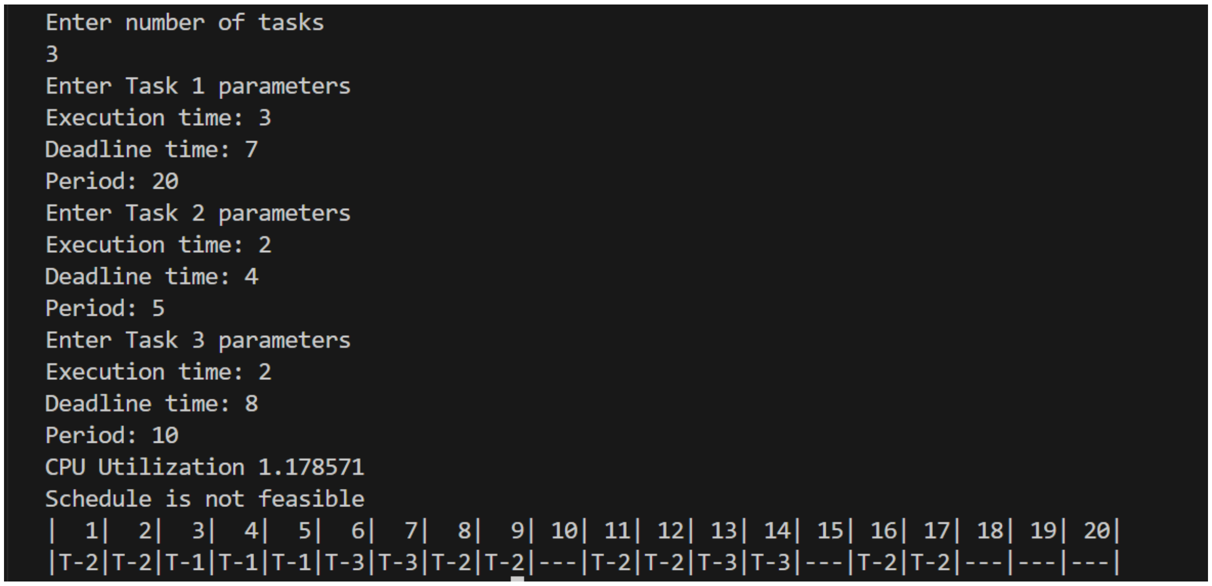
```
#include <stdio.h>
#include <malloc.h>

#define arrival 0
#define execution 1
#define deadline 2
#define period 3
#define abs_arrival 4
#define execution_copy 5
#define abs_deadline 6

typedef struct
{
    int T[7], instance, alive;

} task;

#define IDLE_TASK_ID 1023
#define ALL 1
#define CURRENT 0

void get_tasks(task *t1, int n);
int hyperperiod_calc(task *t1, int n);
float cpu_util(task *t1, int n);
int gcd(int a, int b);
int lcm(int *a, int n);
int sp_interrupt(task *t1, int tmr, int n);
int min(task *t1, int n, int p);
void update_abs_arrival(task *t1, int n, int k, int all);
void update_abs_deadline(task *t1, int n, int all);
void copy_execution_time(task *t1, int n, int all);

int timer = 0;

int main()
{
    task *t;
    int n, hyper_period, active_task_id;
    float cpu_utilization;
    printf("Enter number of tasks\n");
    scanf("%d", &n);
    t = (task *)malloc(n * sizeof(task));
    get_tasks(t, n);
    cpu_utilization = cpu_util(t, n);
    printf("CPU Utilization %f\n", cpu_utilization);

    if (cpu_utilization < 1)
        printf("Tasks can be scheduled\n");
    else
        printf("Schedule is not feasible\n");

    hyper_period = hyperperiod_calc(t, n);
    copy_execution_time(t, n, ALL);
    update_abs_arrival(t, n, 0, ALL);
    update_abs_deadline(t, n, ALL);
```

```
while (timer < hyper_period)
{
    ++timer;
    if (timer < 10)
        printf("|  %d", timer);
    else
        printf("| %d", timer);
}
printf("|\n");

timer = 0;
while (timer < hyper_period)
{

    if (sp_interrupt(t, timer, n))
    {
        active_task_id = min(t, n, abs_deadline);
    }

    if (active_task_id == IDLE_TASK_ID)
    {
        printf("|Idl");
    }

    if (active_task_id != IDLE_TASK_ID)
    {
            if (t[active_task_id].T[execution_copy] != 0)
            {
                t[active_task_id].T[execution_copy]--;
                printf("|T-%d", active_task_id + 1);
            }

            if (t[active_task_id].T[execution_copy] == 0)
            {
                t[active_task_id].instance++;
                t[active_task_id].alive = 0;
                copy_execution_time(t, active_task_id, CURRENT);
                update_abs_arrival(t, active_task_id, t[active_task_id].instance, CURRENT);
                update_abs_deadline(t, active_task_id, CURRENT);
                active_task_id = min(t, n, abs_deadline);
            }
        }
        ++timer;
    }
    printf("|\n");
    free(t);
    return 0;
}

void get_tasks(task *t1, int n)
{
    int i = 0;
    while (i < n)
    {
        printf("Enter Task %d parameters\n", i + 1);
        t1->T[arrival] = 0;
        printf("Execution time: ");
        scanf("%d", &t1->T[execution]);
        printf("Deadline time: ");
        scanf("%d", &t1->T[deadline]);
        printf("Period: ");
        scanf("%d", &t1->T[period]);
        t1->T[abs_arrival] = 0;
        t1->T[execution_copy] = 0;
        t1->T[abs_deadline] = 0;
        t1->instance = 0;
        t1->alive = 0;
        t1++;
        i++;
    }
}

int hyperperiod_calc(task *t1, int n)
{
    int i = 0, ht, a[10];
    while (i < n)

    {
        a[i] = t1->T[period];
        t1++;
        i++;
    }
    ht = lcm(a, n);

    return ht;
}

int gcd(int a, int b)
{
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
}

int lcm(int *a, int n)
{
    int res = 1, i;
    for (i = 0; i < n; i++)
    {
        res = res * a[i] / gcd(res, a[i]);
    }
    return res;
}

int sp_interrupt(task *t1, int tmr, int n)
{
    int i = 0, n1 = 0, a = 0;
    task *t1_copy;
    t1_copy = t1;
```

```
while (i < n)
{
    if (tmr == t1->T[abs_arrival])
    {
        t1->alive = 1;
        a++;
    }
    t1++;
    i++;
}

t1 = t1_copy;
i = 0;

while (i < n)
{
    if (t1->alive == 0)
        n1++;
    t1++;
    i++;
}

if (n1 == n || a != 0)
{
    return 1;
}

return 0;
```

```
}

void update_abs_deadline(task *t1, int n, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
        {
            t1->T[abs_deadline] = t1->T[deadline] + t1->T[abs_arrival];
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[abs_deadline] = t1->T[deadline] + t1->T[abs_arrival];
    }
}

void update_abs_arrival(task *t1, int n, int k, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
        {
            t1->T[abs_arrival] = t1->T[arrival] + k * (t1->T[period]);
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[abs_arrival] = t1->T[arrival] + k * (t1->T[period]);
    }
}

void copy_execution_time(task *t1, int n, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
        {
            t1->T[execution_copy] = t1->T[execution];
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[execution_copy] = t1->T[execution];
    }
}

int min(task *t1, int n, int p)
{
    int i = 0, min = 0x7FFF, task_id = IDLE_TASK_ID;
    while (i < n)
    {
        if (min > t1->T[p] && t1->alive == 1)
        {
            min = t1->T[p];
            task_id = i;
        }
        t1++;
        i++;
    }
    return task_id;
}

float cpu_util(task *t1, int n)
{
    int i = 0;
    float cu = 0;
    while (i < n)
    {
        cu = cu + (float)t1->T[execution] / (float)t1->T[deadline];
        t1++;
        i++;
    }
    return cu;
}
```

OUTPUT:



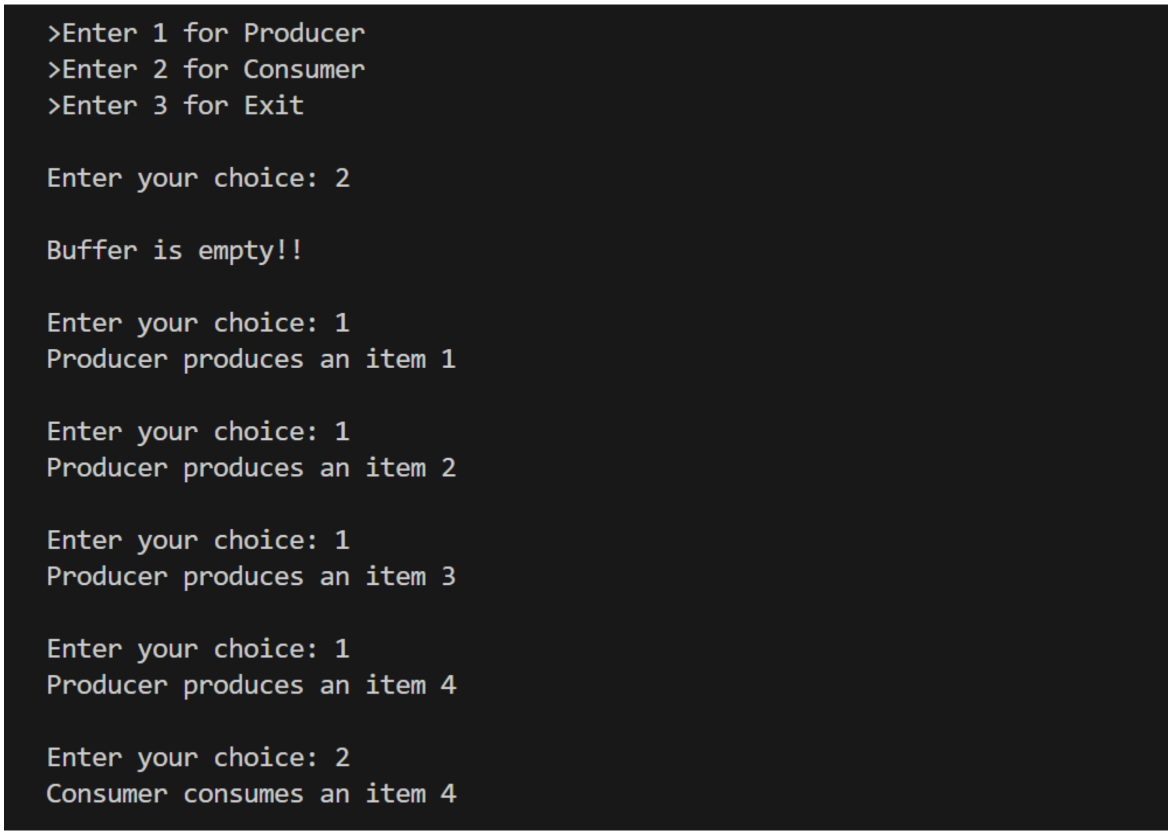
```
Enter number of tasks
3
Enter Task 1 parameters
Execution time: 3
Deadline time: 7
Period: 20
Enter Task 2 parameters
Execution time: 2
Deadline time: 4
Period: 5
Enter Task 3 parameters
Execution time: 2
Deadline time: 8
Period: 10
CPU Utilization 1.178571
Schedule is not feasible
|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20|
|T-2|T-2|T-1|T-1|T-1|T-3|T-3|T-2|T-2|---|T-2|T-2|T-3|T-3|---|T-2|T-2|---|---|---|
```

PROGRAM-5

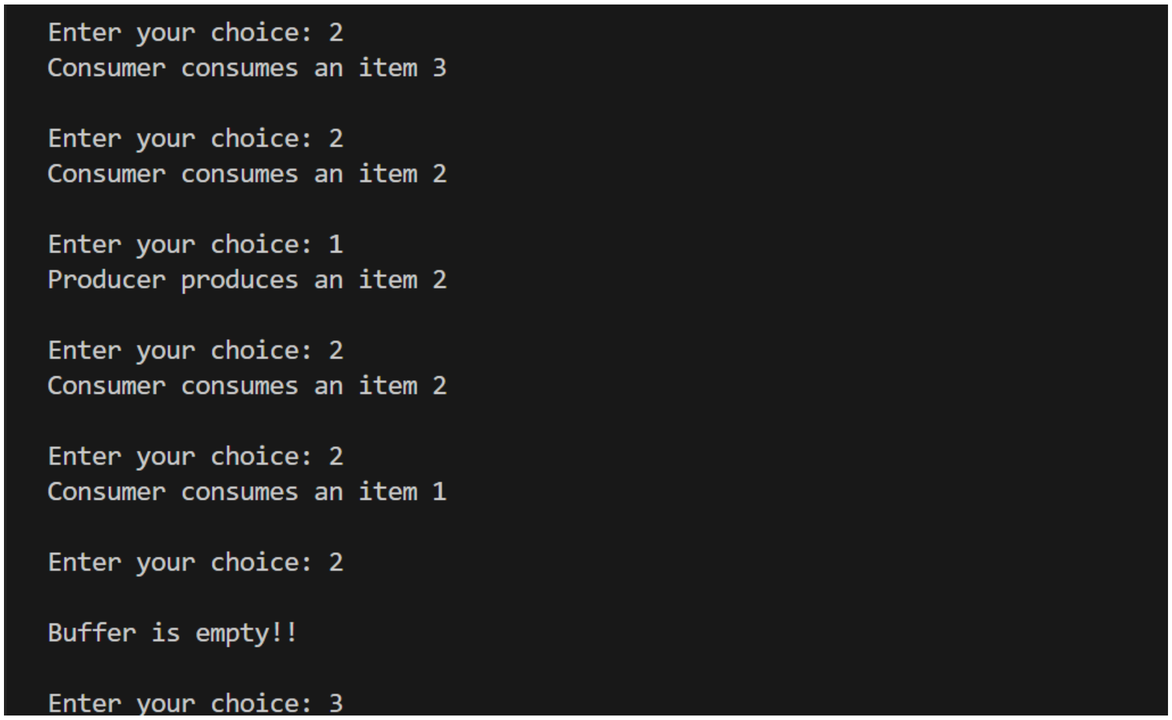
Write a C program to simulate producer-consumer problem using semaphores.

```
#include <stdio.h>

#include <conio.h>

#include <stdlib.h>


int mutex = 1;

int full = 0;

int empty = 10;

int count = 0;


int wait(int s)

{

    while (s <= 0)

        ;

    s--;

    return s;

}


int signal(int s)

{

    s++;

    return s;

}


void producer()

{
    empty = wait(empty);
    mutex = wait(mutex);
    count++;
    printf("Producer produces an item %d\n", count);
    mutex = signal(mutex);
    full = signal(full);
}

void consumer()
{
    full = wait(full);
    mutex = wait(mutex);
    printf("Consumer consumes an item %d\n", count);
    count--;
    mutex = signal(mutex);
    empty = signal(empty);
}
void main()
{
    int choice;
    printf("\n>Enter 1 for Producer\n>Enter 2 for Consumer\n>Enter 3 for Exit\n");
    while (1)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            if (empty == 0)
            {
                printf("\nBuffer is full!!\n");
            }
            else
            {
                producer();
            }
            break;
        case 2:
            if (full == 0)
            {
                printf("\nBuffer is empty!!\n");
            }
            else
            {
                consumer();
            }
            break;
        case 3:
            exit(0);
            break;
        default:
            printf("Invalid choice\n");
        }
    }
}
```

OUTPUT:



```
>Enter 1 for Producer
>Enter 2 for Consumer
>Enter 3 for Exit

Enter your choice: 2

Buffer is empty!!

Enter your choice: 1
Producer produces an item 1

Enter your choice: 1
Producer produces an item 2

Enter your choice: 1
Producer produces an item 3

Enter your choice: 1
Producer produces an item 4

Enter your choice: 2
Consumer consumes an item 4
```



```
Enter your choice: 2
Consumer consumes an item 3

Enter your choice: 2
Consumer consumes an item 2

Enter your choice: 1
Producer produces an item 2

Enter your choice: 2
Consumer consumes an item 2

Enter your choice: 2
Consumer consumes an item 1

Enter your choice: 2

Buffer is empty!!

Enter your choice: 3
```

PROGRAM-6

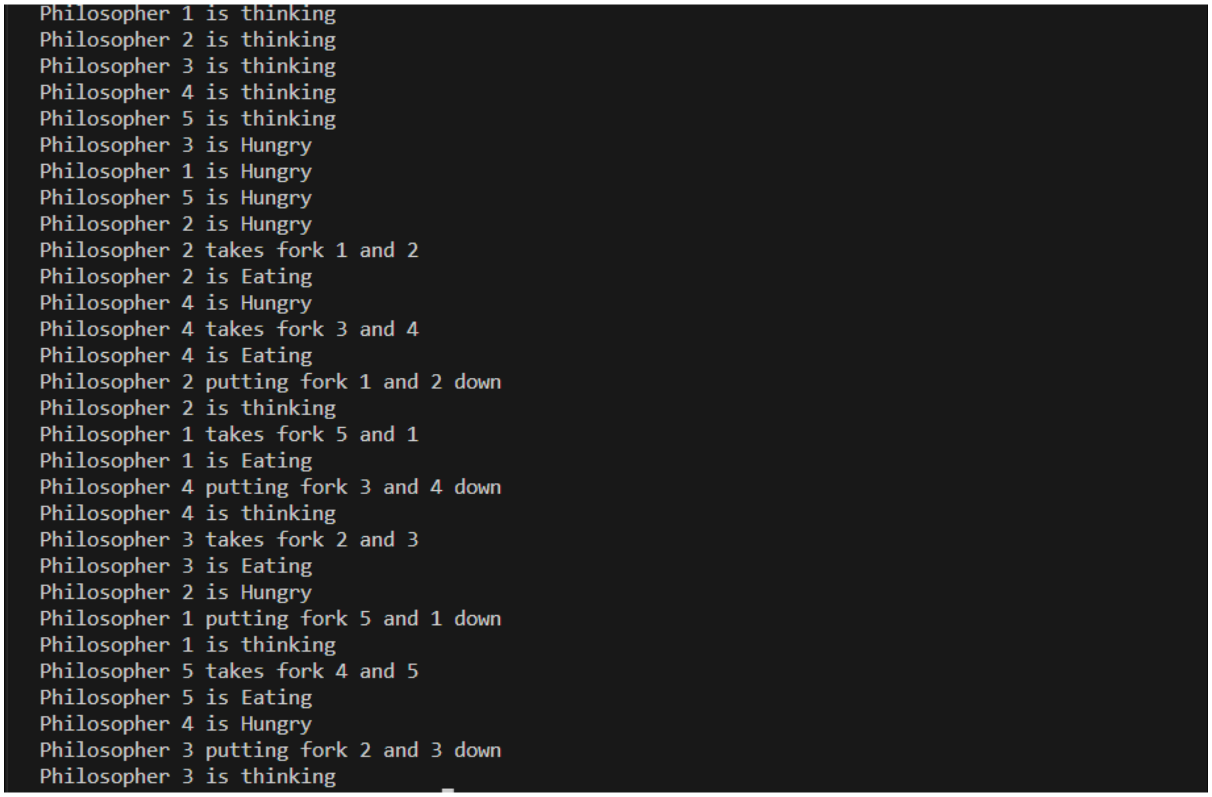
Write a C program to simulate the concept of Dining-Philosophers problem.

```
#include <pthread.h>

#include <semaphore.h>

#include <stdio.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[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]);
    }
}

void take_fork(int phnum)
{

    sem_wait(&mutex);
    state[phnum] = HUNGRY;
    printf("Philosopher %d is Hungry\n", phnum + 1);
    test(phnum);
    sem_post(&mutex);
    sem_wait(&S[phnum]);
    sleep(1);
}

void put_fork(int phnum)
{

    sem_wait(&mutex);
    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 *philosopher(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];
    sem_init(&mutex, 0, 1);

    for (i = 0; i < N; i++)
        sem_init(&S[i], 0, 0);

    for (i = 0; i < N; i++)
    {
        pthread_create(&thread_id[i], NULL, philosopher, &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 3 is Hungry
Philosopher 1 is Hungry
Philosopher 5 is Hungry
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 is Hungry
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 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
```

PROGRAM-7

Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance.

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

int main()
{
    int N, M = 3, ind = 0;
    printf("\nEnter the number of processess: ");
    scanf("%d", &N);
    printf("Enter the number of resources: ");
    scanf("%d", &M);

    int alloc[N][M], max[N][M], need[N][M], finished[N], ans[N], avail[M];

    printf("\nEnter allocated resources\n");
    for (int i = 0; i < N; i++)
    {
        printf("For Process %d: ", i);
        for (int j = 0; j < M; j++)
        {
            scanf("%d", &alloc[i][j]);
        }
    }

    printf("\nEnter Maximum resources\n");
    for (int i = 0; i < N; i++)
```

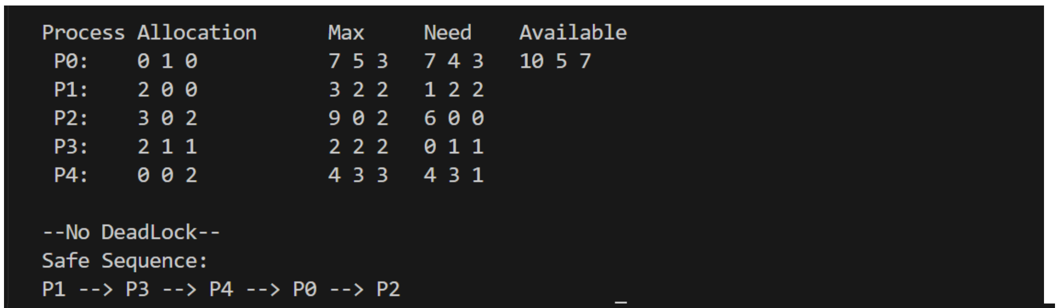
```
{
    printf("For Process %d: ", i);
    for (int j = 0; j < M; j++)
    {
        scanf("%d", &max[i][j]);
    }
}

printf("\nEnter available resources\n");
for (int i = 0; i < M; i++)
{
    scanf("%d", &avail[i]);
}

for (int i = 0; i < N; i++)
{
    finished[i] = 0;
}

for (int i = 0; i < N; i++)
{
    for (int j = 0; j < M; j++)
    {
        need[i][j] = max[i][j] - alloc[i][j];
    }
}

for (int k = 0; k < N; k++)
```

```
{
    for (int i = 0; i < N; i++)
    {
        if (finished[i] == 0)
        {
            int flag = 0;
            for (int j = 0; j < M; j++)
            {
                if (need[i][j] > avail[j])
                {
                    flag = 1;
                    break;
                }
            }

            if (flag == 0)
            {
                ans[ind++] = i;
                for (int p = 0; p < M; p++)
                {
                    avail[p] += alloc[i][p];
                }
                finished[i] = 1;
            }
        }
    }
}
```

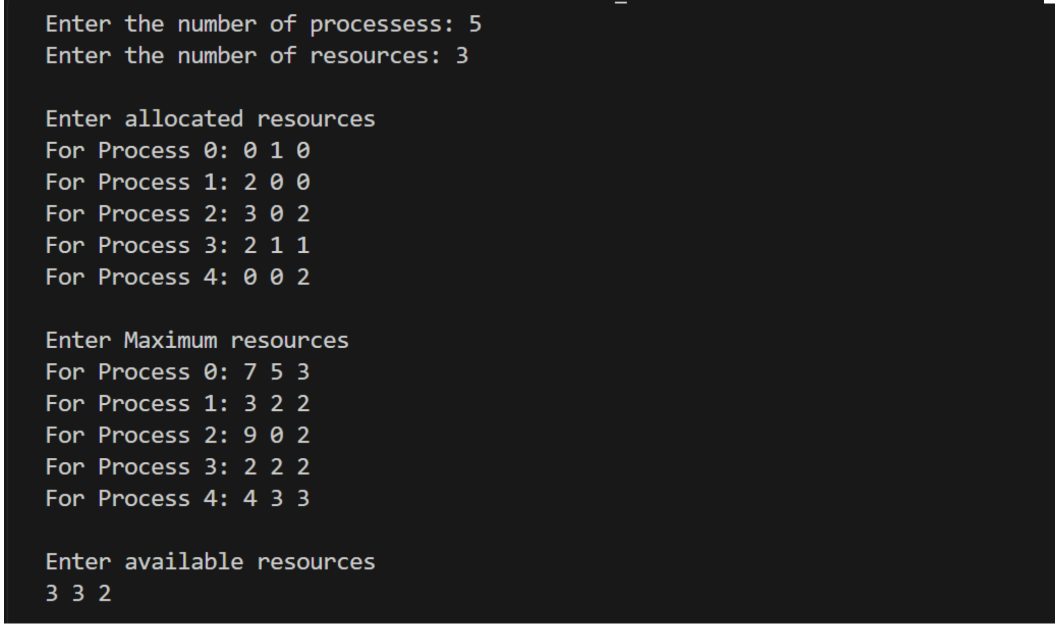
```
printf("\nProcess\tAllocation\tMax\tNeed\tAvailable");
for (int i = 0; i < N; i++)
{
    printf("\n P%d: \t", i);

    for (int j = 0; j < M; j++)
        printf("%d ", alloc[i][j]);
    printf("\t\t");

    for (int j = 0; j < M; j++)
        printf("%d ", max[i][j]);

    printf("\t");

    for (int j = 0; j < M; j++)
        printf("%d ", need[i][j]);

    printf("\t");

    if (i == 0)
    {
        for (int j = 0; j < M; j++)
            printf("%d ", avail[j]);
    }
}

int flag = 1;
for (int i = 0; i < N; i++)
```

```
    {
        if (finished[i] == 0)
        {
            flag = 0;
            printf("\n\nThe System is NOT in safe state(DeadLock Detected)\n");
            break;
        }
    }

    if (flag == 1)
    {
        printf("\n\n--No DeadLock--\nSafe Sequence:\n");
        for (int i = 0; i < N - 1; i++)
        {
            printf("P%d --> ", ans[i]);
        }
        printf("P%d\n", ans[N - 1]);
    }
}
```

OUTPUT:



```
Process Allocation      Max     Need    Available
 P0:    0 1 0          7 5 3   7 4 3   10 5 7
 P1:    2 0 0          3 2 2   1 2 2
 P2:    3 0 2          9 0 2   6 0 0
 P3:    2 1 1          2 2 2   0 1 1
 P4:    0 0 2          4 3 3   4 3 1

--No DeadLock--
Safe Sequence:
P1 --> P3 --> P4 --> P0 --> P2
```



```
Enter the number of processess: 5
Enter the number of resources: 3

Enter allocated resources
For Process 0: 0 1 0
For Process 1: 2 0 0
For Process 2: 3 0 2
For Process 3: 2 1 1
For Process 4: 0 0 2

Enter Maximum resources
For Process 0: 7 5 3
For Process 1: 3 2 2
For Process 2: 9 0 2
For Process 3: 2 2 2
For Process 4: 4 3 3

Enter available resources
3 3 2
```

PROGRAM-8

Write a C program to simulate deadlock detection.

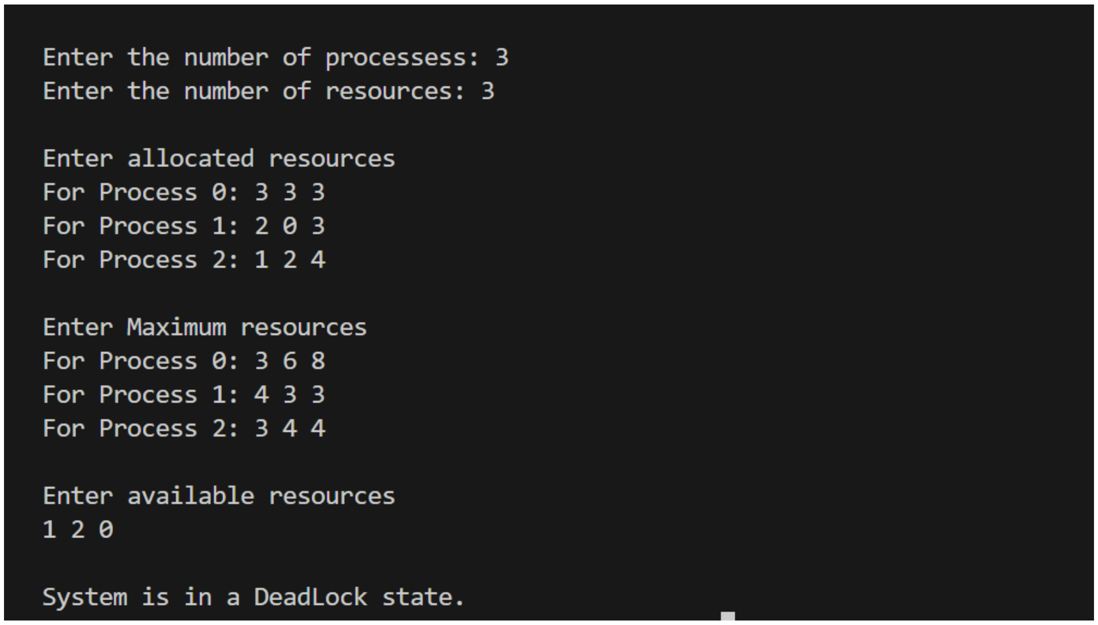
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int N, M, ind = 0;
   printf("\nEnter the number of processess: ");
   scanf("%d", &N);
   printf("Enter the number of resources: ");
   scanf("%d", &M);

   int alloc[N][M], max[N][M], need[N][M], finished[N], ans[N], avail[M];
```

```
printf("\nEnter allocated resources\n");
for (int i = 0; i < N; i++)
{
    printf("For Process %d: ", i);
    for (int j = 0; j < M; j++)
    {
        scanf("%d", &alloc[i][j]);
    }
}

printf("\nEnter Maximum resources\n");
for (int i = 0; i < N; i++)
{
    printf("For Process %d: ", i);
    for (int j = 0; j < M; j++)
    {
        scanf("%d", &max[i][j]);
    }
}

printf("\nEnter available resources\n");
for (int i = 0; i < M; i++)
{
    scanf("%d", &avail[i]);
}

for (int i = 0; i < N; i++)
```

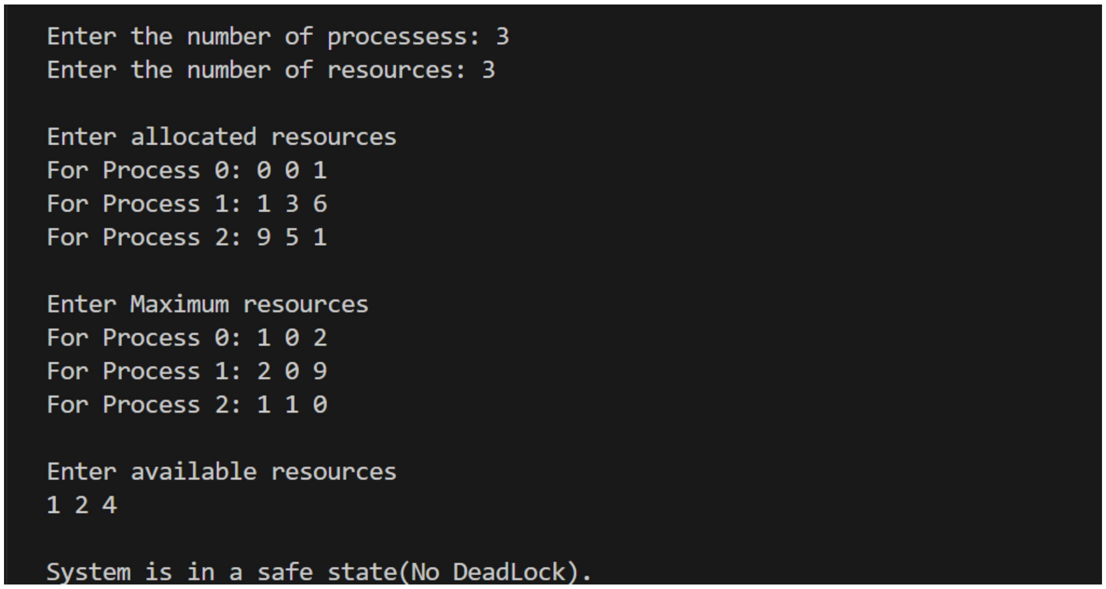
```
{
    finished[i] = 0;
}

for (int i = 0; i < N; i++)
{
    for (int j = 0; j < M; j++)
    {
        need[i][j] = max[i][j] - alloc[i][j];
    }
}

for (int k = 0; k < N; k++)
{
    for (int i = 0; i < N; i++)
    {
        if (finished[i] == 0)
        {
            int flag = 0;
            for (int j = 0; j < M; j++)
            {
                if (need[i][j] > avail[j])
                {
                    flag = 1;
                    break;
                }
            }
            if (flag == 0)
            {
                ans[ind++] = i;
                for (int p = 0; p < M; p++)
                {
                    avail[p] += alloc[i][p];
                }
                finished[i] = 1;
            }
        }
    }
}

int flag = 1;
for (int i = 0; i < N; i++)
{
    if (finished[i] == 0)
    {
        flag = 0;
        printf("\nSystem is in a DeadLock state.\n");
        break;
    }
}

if (flag == 1)
{
    printf("\nSystem is in a safe state(No DeadLock).\n");
}
```

}

OUTPUT:



```
Enter the number of processess: 3
Enter the number of resources: 3

Enter allocated resources
For Process 0: 3 3 3
For Process 1: 2 0 3
For Process 2: 1 2 4

Enter Maximum resources
For Process 0: 3 6 8
For Process 1: 4 3 3
For Process 2: 3 4 4

Enter available resources
1 2 0

System is in a DeadLock state.
```



```
Enter the number of processess: 3
Enter the number of resources: 3

Enter allocated resources
For Process 0: 0 0 1
For Process 1: 1 3 6
For Process 2: 9 5 1

Enter Maximum resources
For Process 0: 1 0 2
For Process 1: 2 0 9
For Process 2: 1 1 0

Enter available resources
1 2 4

System is in a safe state(No DeadLock).
```

PROGRAM-9

Write a C program to simulate the following contiguous memory allocation techniques

a) Worst-fit

b) Best-fit

c) First-fit

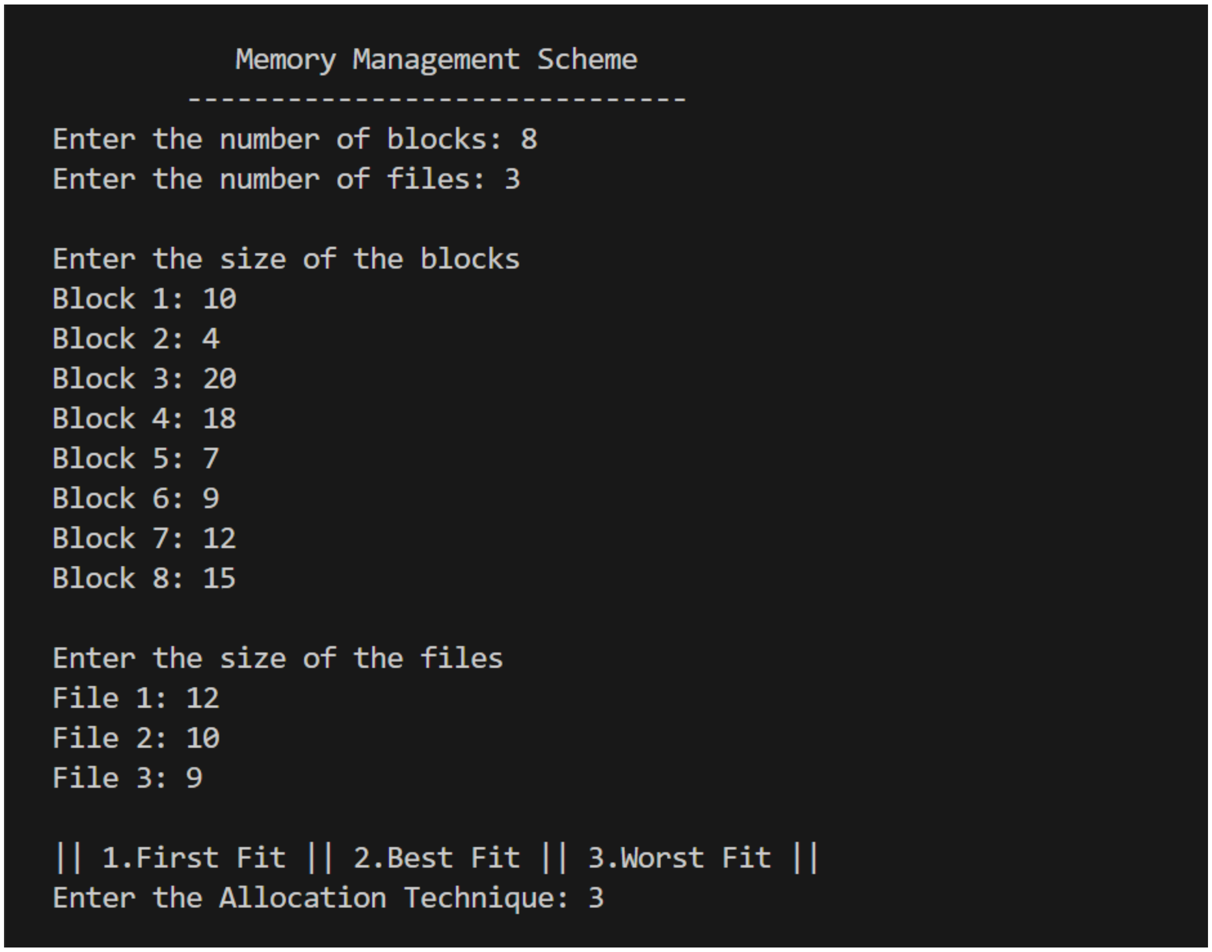
```
#include <stdio.h>
#define max 25

int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0, lowest = 10000, ch;
static int bf[max], ff[max];

void firstfit();
void bestfit();
void worstfit();

void main()
{
    printf("\n\t   Memory Management Scheme\n");
    printf("\t------------------------------");

    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("\nEnter the size of the files\n");
for (i = 1; i <= nf; i++)
{
    printf("File %d: ", i);
    scanf("%d", &f[i]);
}

printf("\n1.First Fit || 2.Best Fit || 3.Worst Fit ||\nEnter the Allocation Technique: ");
scanf("%d", &ch);
switch (ch)
{
case 1:
    firstfit();
    break;
case 2:
    bestfit();
    break;
case 3:
    worstfit();
    break;
default:
    printf("Invalid choice");
}
```

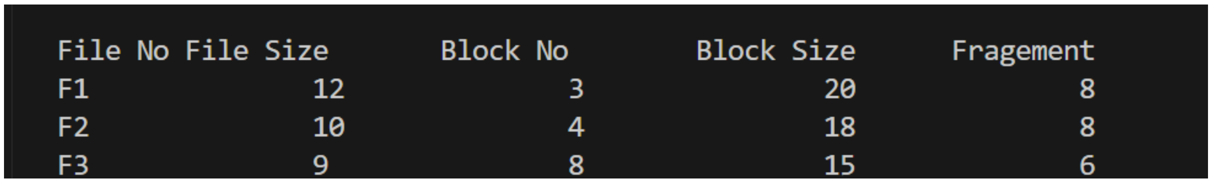
```
}

void firstfit()
{
    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("\nF%d\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}

void bestfit()
{
    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("\nF%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}

void worstfit()
{
    for (i = 1; i <= nf; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1)
            {
                temp = b[j] - f[i];
                if (temp >= 0)
                    if (highest < temp)
                    {
                        ff[i] = j;
                        highest = temp;
                    }
            }
        }
        frag[i] = highest;
        bf[ff[i]] = 1;
        highest = 0;
    }
    printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragement");
    for (i = 1; i <= nf; i++)
        printf("\nF%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
```

OUTPUT:

1) Worst-fit

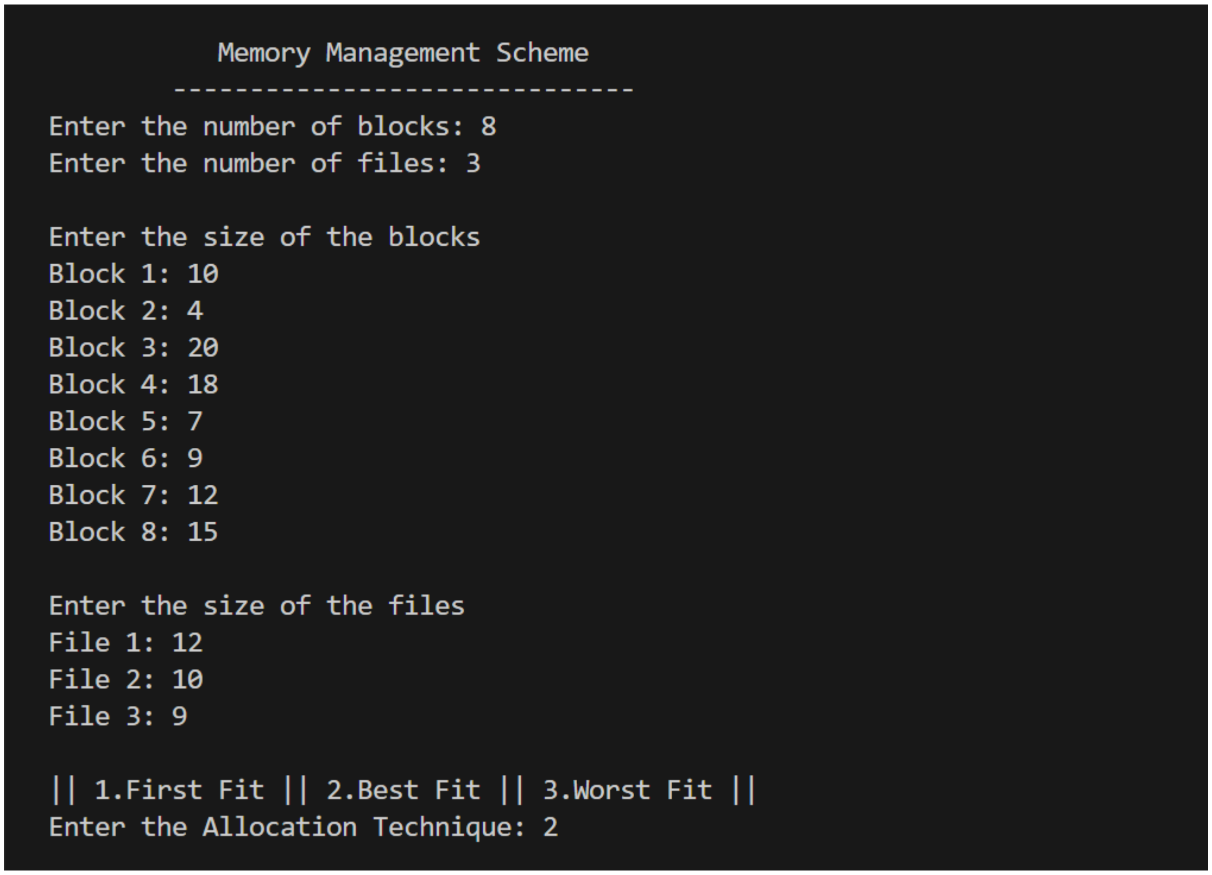


```
            Memory Management Scheme
        ------------------------------
Enter the number of blocks: 8
Enter the number of files: 3

Enter the size of the blocks
Block 1: 10
Block 2: 4
Block 3: 20
Block 4: 18
Block 5: 7
Block 6: 9
Block 7: 12
Block 8: 15

Enter the size of the files
File 1: 12
File 2: 10
File 3: 9

|| 1.First Fit || 2.Best Fit || 3.Worst Fit ||
Enter the Allocation Technique: 3
```

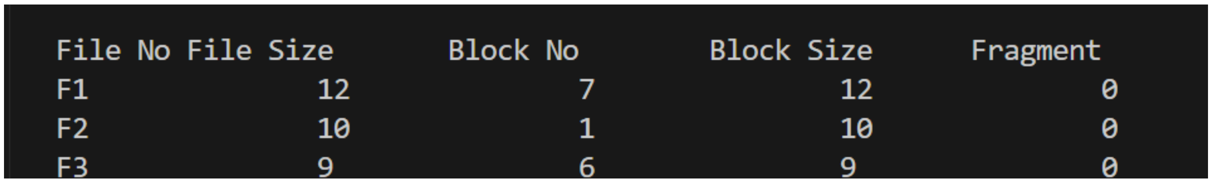


```
File No File Size       Block No        Block Size      Fragement
F1              12             3                20              8
F2              10             4                18              8
F3               9             8                15              6
```

2) Best-Fit

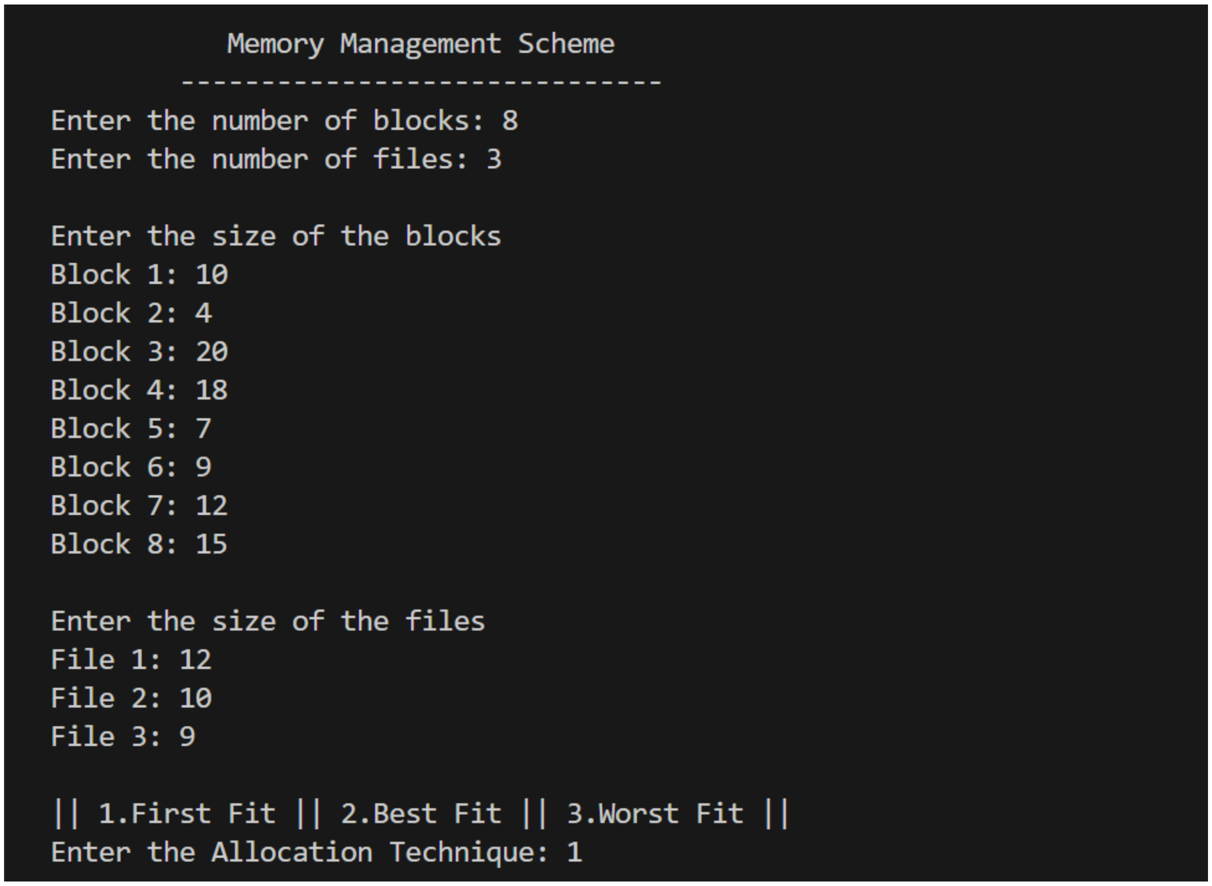


```
             Memory Management Scheme
          ------------------------------
Enter the number of blocks: 8
Enter the number of files: 3

Enter the size of the blocks
Block 1: 10
Block 2: 4
Block 3: 20
Block 4: 18
Block 5: 7
Block 6: 9
Block 7: 12
Block 8: 15

Enter the size of the files
File 1: 12
File 2: 10
File 3: 9

|| 1.First Fit || 2.Best Fit || 3.Worst Fit ||
Enter the Allocation Technique: 2
```

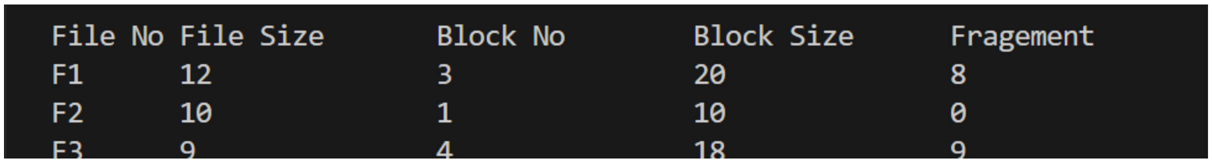


```
File No File Size        Block No         Block Size        Fragment
F1                12                7               12               0
F2                10                1               10               0
F3                9                 6               9                0
```

3) First-Fit



```
             Memory Management Scheme
          ------------------------------
Enter the number of blocks: 8
Enter the number of files: 3

Enter the size of the blocks
Block 1: 10
Block 2: 4
Block 3: 20
Block 4: 18
Block 5: 7
Block 6: 9
Block 7: 12
Block 8: 15

Enter the size of the files
File 1: 12
File 2: 10
File 3: 9

|| 1.First Fit || 2.Best Fit || 3.Worst Fit ||
Enter the Allocation Technique: 1
```



```
File No File Size       Block No        Block Size      Fragement
F1      12              3               20              8
F2      10              1               10              0
F3      9               4               18              9
```

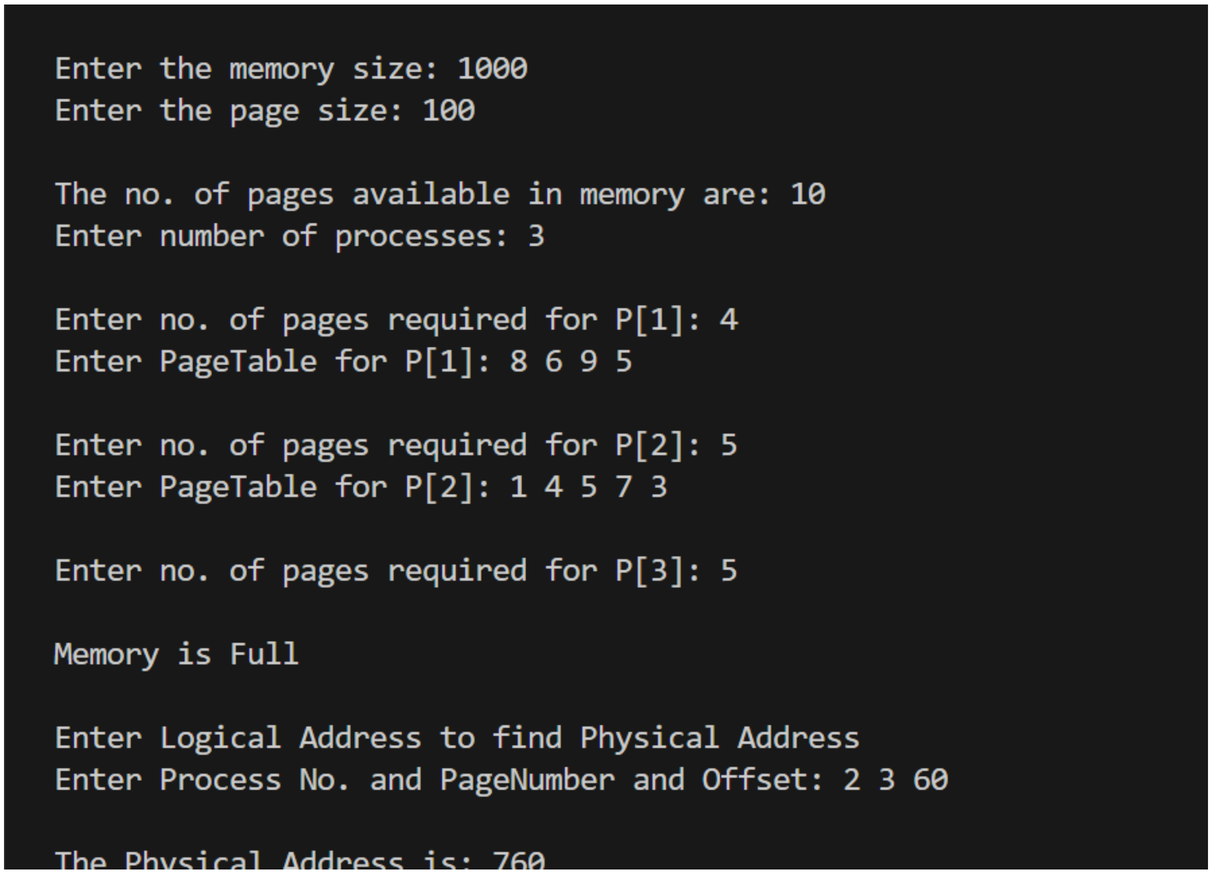
PROGRAM-10

Write a C program to simulate paging technique of memory management.

```
#include <stdio.h>
void main()
{
   int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
   int s[10], fno[10][20];

   printf("\nEnter the memory size: ");
   scanf("%d", &ms);

   printf("Enter the page size: ");
   scanf("%d", &ps);

   nop = ms / ps;
   printf("\nThe no. of pages available in memory are: %d ", nop);

   printf("\nEnter number of processes: ");
   scanf("%d", &np);
   rempages = nop;
   for (i = 1; i <= np; i++)

   {
      printf("\nEnter no. of pages required for P[%d]: ", i);
      scanf("%d", &s[i]);

      if (s[i] > rempages)
```

```
        {
            printf("\nMemory is Full\n");
            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("\nEnter Logical Address to find Physical Address ");
    printf("\nEnter Process No. and PageNumber and Offset: ");

    scanf("%d %d %d", &x, &y, &offset);

    if (x > np || y >= s[i] || offset >= ps)
        printf("\nInvalid Process or Page Number or offset\n");
    else
    {
        pa = fno[x][y] * ps + offset;
        printf("\nThe Physical Address is: %d", pa);
    }
}
```

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 6 9 5

Enter no. of pages required for P[2]: 5
Enter PageTable for P[2]: 1 4 5 7 3

Enter no. of pages required for P[3]: 5

Memory is Full

Enter Logical Address to find Physical Address
Enter Process No. and PageNumber and Offset: 2 3 60

The Physical Address is: 760
```

PROGRAM-11

Write a C program to simulate page replacement algorithms

a) FIFO

b) LRU

c) Optimal

```
#include <stdio.h>

#define MAX_FRAMES 3
#define MAX_PAGES 20

void fifo(int pages[], int n, int frames)
{
    int frame[frames];
    int front = 0, rear = 0;
    int page_faults = 0;

    for (int i = 0; i < frames; i++)
    {
        frame[i] = -1;
    }

    for (int i = 0; i < n; i++)
    {
        int found = 0;
        for (int j = 0; j < frames; j++)
        {
            if (frame[j] == pages[i])
            {
                found = 1;
                break;
            }
        }

        if (!found)
        {
            frame[rear] = pages[i];
            rear = (rear + 1) % frames;
            page_faults++;
        }

        printf("Page %d: ", pages[i]);
        for (int j = 0; j < frames; j++)
        {
            if (frame[j] == -1)
                printf("- ");
            else
                printf("%d ", frame[j]);
        }
        printf("\n");
    }

    printf("Total Page Faults (FIFO): %d\n", page_faults);
}

void lru(int pages[], int n, int frames)
```

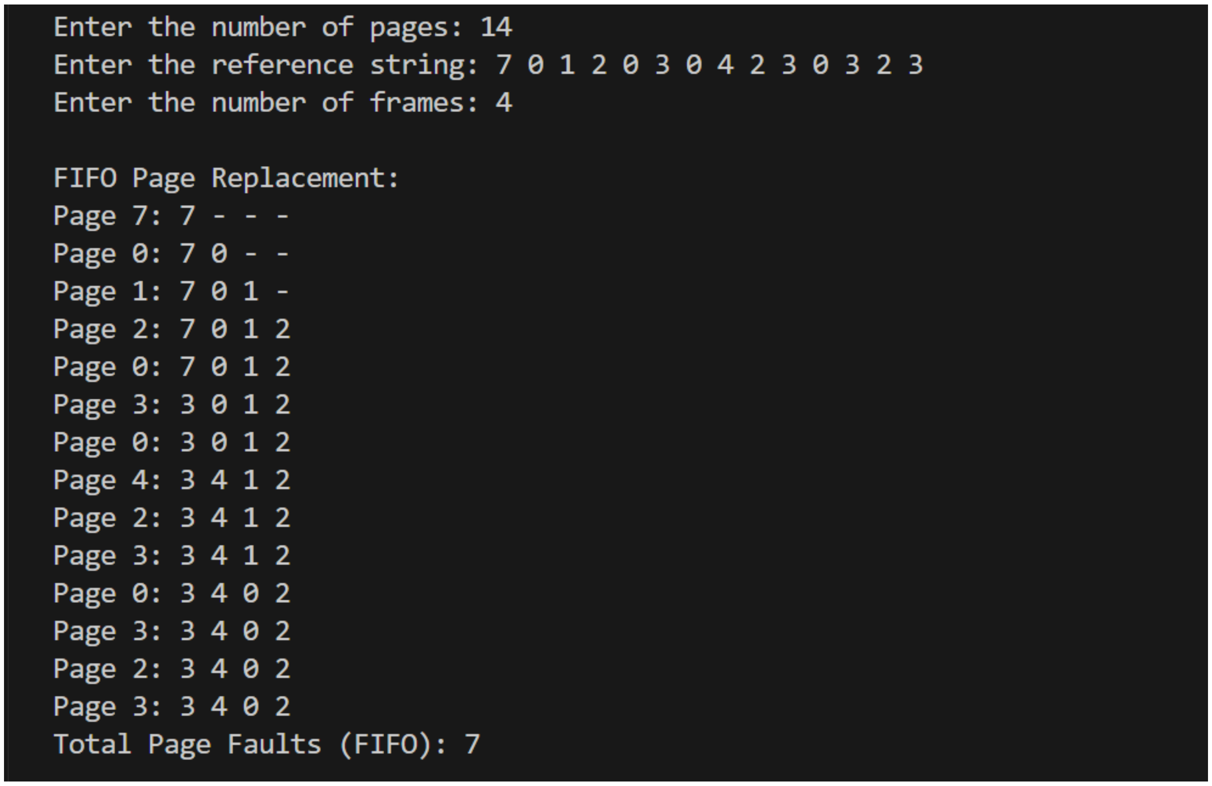
```
{
    int frame[frames];
    int page_faults = 0;
    int used[MAX_PAGES] = {0};

    for (int i = 0; i < frames; i++)
    {
        frame[i] = -1;
    }

    for (int i = 0; i < n; i++)
    {
        int found = 0;
        for (int j = 0; j < frames; j++)
        {
            if (frame[j] == pages[i])
            {
                found = 1;
                used[j] = i;
                break;
            }
        }

        if (!found)
        {
            int min = 0;
            for (int j = 1; j < frames; j++)
            {
                if (used[j] < used[min])
                {
                    min = j;
                }
            }
            frame[min] = pages[i];
            used[min] = i;
            page_faults++;
        }

        printf("Page %d: ", pages[i]);
        for (int j = 0; j < frames; j++)
        {
            if (frame[j] == -1)
                printf("- ");
            else
                printf("%d ", frame[j]);
        }
        printf("\n");
    }

    printf("Total Page Faults (LRU): %d\n", page_faults);
}

void optimal (int pages[], int n, int frames)
{
    int frame[frames];
    int page_faults = 0;
```

```
for (int i = 0; i < frames; i++)
{
    frame[i] = -1;
}

for (int i = 0; i < n; i++)
{
    int found = 0;
    for (int j = 0; j < frames; j++)
    {
        if (frame[j] == pages[i])
        {
            found = 1;
            break;
        }
    }

    if (!found)
    {
        if (i < frames)
        {
            frame[i] = pages[i];
        }
        else
        {
            int max_dist = -1;
            int replace_page = -1;
        for (int j = 0; j < frames; j++)
        {
            int dist = MAX_PAGES;
            for (int k = i + 1; k < n; k++)
            {
                if (pages[k] == frame[j])
                {
                    dist = k - i;
                    break;
                }
            }
            if (dist > max_dist)
            {
                max_dist = dist;
                replace_page = j;
            }
        }
        frame[replace_page] = pages[i];
    }
    page_faults++;
}

printf("Page %d: ", pages[i]);
for (int j = 0; j < frames; j++)
{
    if (frame[j] == -1)
        printf("- ");
    else
```

```
                printf("%d ", frame[j]);
            }
            printf("\n");
        }

        printf("Total Page Faults (Optimal): %d\n", page_faults);
    }

    int main()
    {
        int pages[MAX_PAGES];
        int n, frames;

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

        printf("Enter the reference string: ");
        for (int i = 0; i < n; i++)
        {
            scanf("%d", &pages[i]);
        }

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

        printf("\nFIFO Page Replacement:\n");
        fifo(pages, n, frames);
    printf("\nLRU Page Replacement:\n");
    lru(pages, n, frames);


    printf("\nOptimal Page Replacement:\n");
    optimal(pages, n, frames);


    return 0;
}
```

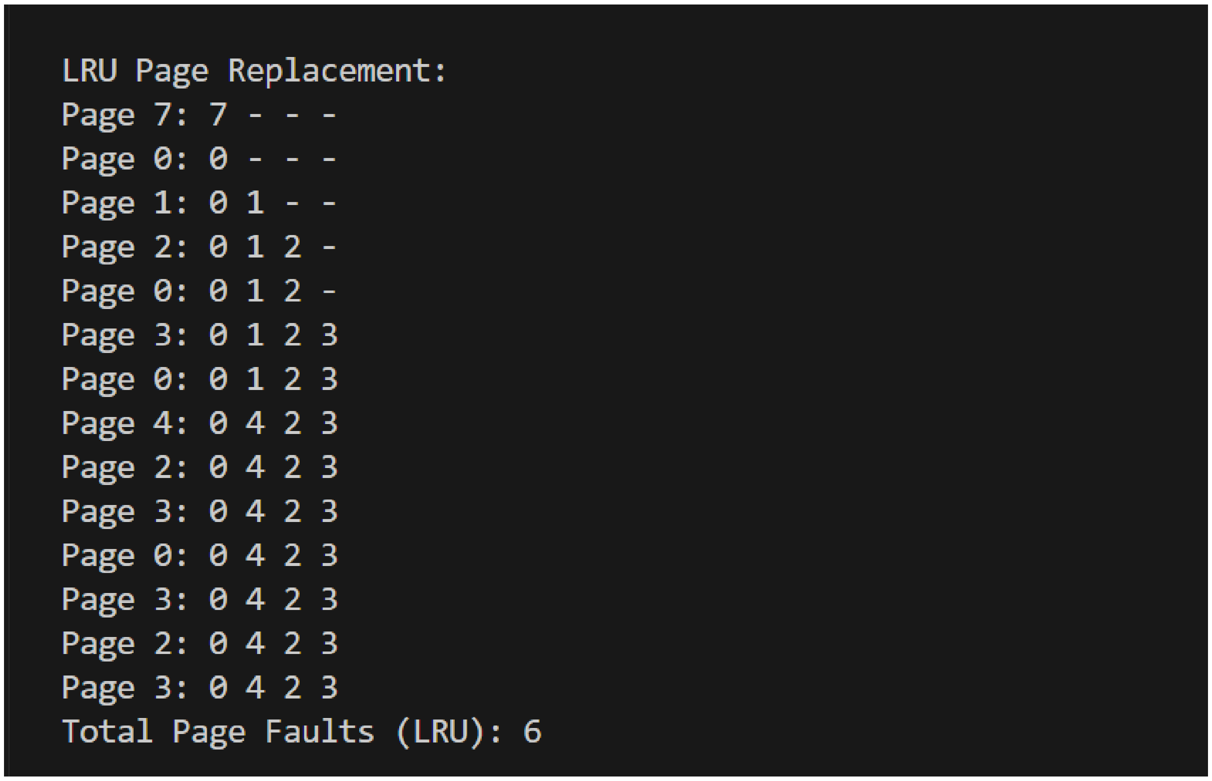
OUTPUT:

a) FIFO



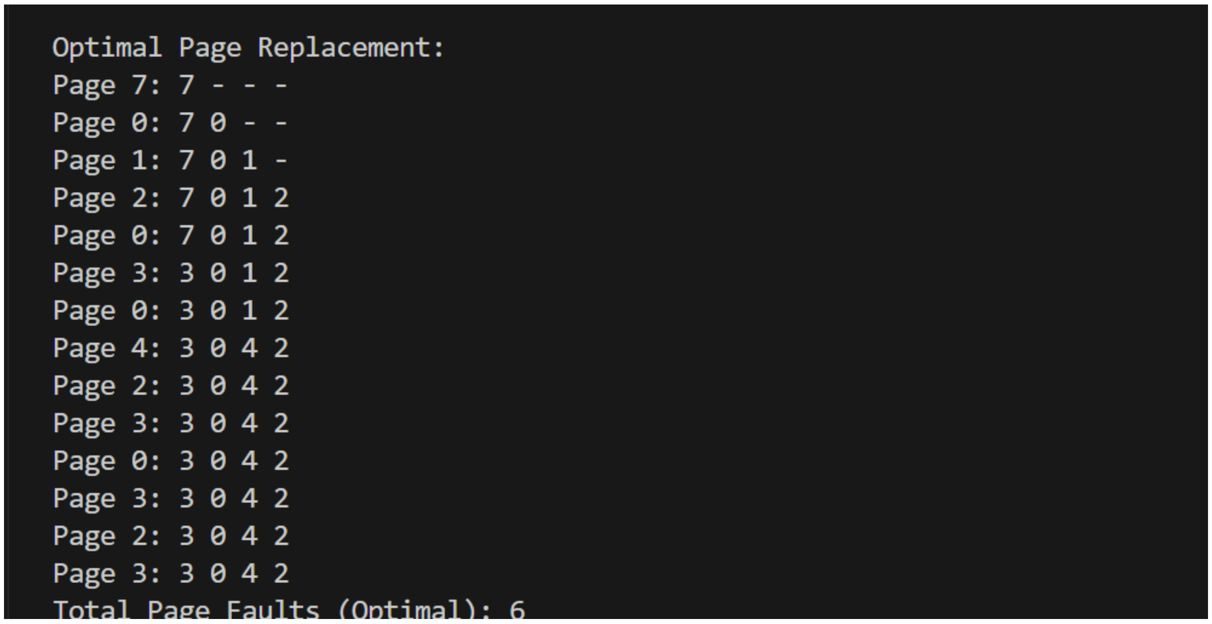
```
Enter the number of pages: 14
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 3
Enter the number of frames: 4

FIFO Page Replacement:
Page 7: 7 - - -
Page 0: 7 0 - -
Page 1: 7 0 1 -
Page 2: 7 0 1 2
Page 0: 7 0 1 2
Page 3: 3 0 1 2
Page 0: 3 0 1 2
Page 4: 3 4 1 2
Page 2: 3 4 1 2
Page 3: 3 4 1 2
Page 0: 3 4 0 2
Page 3: 3 4 0 2
Page 2: 3 4 0 2
Page 3: 3 4 0 2
Total Page Faults (FIFO): 7
```

b) LRU



```
LRU Page Replacement:
Page 7: 7 - - -
Page 0: 0 - - -
Page 1: 0 1 - -
Page 2: 0 1 2 -
Page 0: 0 1 2 -
Page 3: 0 1 2 3
Page 0: 0 1 2 3
Page 4: 0 4 2 3
Page 2: 0 4 2 3
Page 3: 0 4 2 3
Page 0: 0 4 2 3
Page 3: 0 4 2 3
Page 2: 0 4 2 3
Page 3: 0 4 2 3
Total Page Faults (LRU): 6
```

c) Optimal



```
Optimal Page Replacement:
Page 7: 7 - - -
Page 0: 7 0 - -
Page 1: 7 0 1 -
Page 2: 7 0 1 2
Page 0: 7 0 1 2
Page 3: 3 0 1 2
Page 0: 3 0 1 2
Page 4: 3 0 4 2
Page 2: 3 0 4 2
Page 3: 3 0 4 2
Page 0: 3 0 4 2
Page 3: 3 0 4 2
Page 2: 3 0 4 2
Page 3: 3 0 4 2
Total Page Faults (Optimal): 6
```

PROGRAM-14

Write a C program to simulate disk scheduling algorithms

a) FCFS

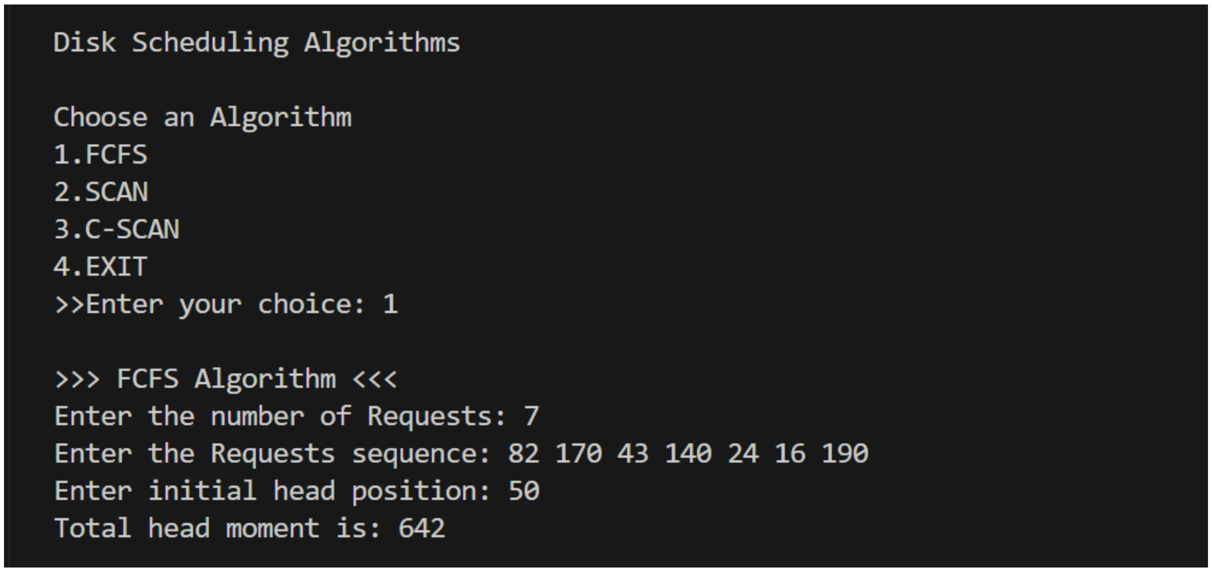
b) SCAN

c) C-SCAN

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

void FCFS()
{
    int RQ[100], i, n, TotalHeadMoment = 0, initial;
    printf("\n>>> FCFS Algorithm <<<");
    printf("\nEnter the number of Requests: ");
    scanf("%d", &n);
    printf("Enter the Requests sequence: ");
    for (i = 0; i < n; i++)
        scanf("%d", &RQ[i]);

    printf("Enter initial head position: ");
    scanf("%d", &initial);

    for (i = 0; i < n; i++)
    {
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
        initial = RQ[i];
    }
    printf("Total head moment is: %d", TotalHeadMoment);
}


void SCAN()
{
    int RQ[100], i, j, n, TotalHeadMoment = 0, initial, size, move;
    printf("\n>>> SCAN Algorithm <<<");
    printf("\nEnter the number of Requests: ");
    scanf("%d", &n);
    printf("Enter the Requests sequence: ");
    for (i = 0; i < n; i++)
        scanf("%d", &RQ[i]);


    printf("Enter initial head position: ");
    scanf("%d", &initial);
    printf("Enter total disk size: ");
    scanf("%d", &size);
    printf(">>Choose the head movement direction\n>0.Towards the smaller value\n>1.Towards
the larger value\n>>: ");
    scanf("%d", &move);


    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (RQ[j] > RQ[j + 1])
            {
                int temp;
                temp = RQ[j];
                RQ[j] = RQ[j + 1];
                RQ[j + 1] = temp;
            }
        }
    }

    int index;
    for (i = 0; i < n; i++)
    {
        if (initial < RQ[i])
        {
            index = i;
            break;
        }
    }
    if (move == 1)
    {
        for (i = index; i < n; i++)
        {
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
        TotalHeadMoment = TotalHeadMoment + abs(size - RQ[i - 1] - 1);
        initial = size - 1;
        for (i = index - 1; i >= 0; i--)
        {
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
    }
    else
    {
        for (i = index - 1; i >= 0; i--)
        {
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i + 1] - 0);
        initial = 0;
        for (i = index; i < n; i++)
        {
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
    }

    printf("Total head movement is: %d", TotalHeadMoment);
}
void C_SCAN()
{
    int RQ[100], i, j, n, TotalHeadMoment = 0, initial, size, move;

    printf("\n>>> C-SCAN Algorithm <<<");
    printf("\nEnter the number of Requests: ");
    scanf("%d", &n);
    printf("Enter the Requests sequence: ");
    for (i = 0; i < n; i++)
        scanf("%d", &RQ[i]);

    printf("Enter initial head position: ");
    scanf("%d", &initial);
    printf("Enter total disk size: ");
    scanf("%d", &size);
    printf(">>Choose the head movement direction\n>0.Towards the smaller value\n>1.Towards
the larger value\n>>: ");
    scanf("%d", &move);

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (RQ[j] > RQ[j + 1])
            {
                int temp;
                temp = RQ[j];
                RQ[j] = RQ[j + 1];
                RQ[j + 1] = temp;
            }
        }
    }

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

    if (move == 1)
    {
        for (i = index; i < n; i++)
        {
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
        TotalHeadMoment = TotalHeadMoment + abs(size - RQ[i - 1] - 1);
        TotalHeadMoment = TotalHeadMoment + abs(size - 1 - 0);
        initial = 0;
        for (i = 0; i < index; i++)
        {
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
    }
    else
    {
        for (i = index - 1; i >= 0; i--)
        {
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i + 1] - 0);
        TotalHeadMoment = TotalHeadMoment + abs(size - 1 - 0);
        initial = size - 1;
        for (i = n - 1; i >= index; i--)
        {
            TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
            initial = RQ[i];
        }
    }
    printf("Total head movement is: %d", TotalHeadMoment);
}

void main()
{
    int ch;
    printf("\nDisk Scheduling Algorithms");
    while (1)
    {
        printf("\n\nChoose an Algorithm\n");
        printf("1.FCFS\n2.SCAN\n3.C-SCAN\n4.EXIT");
        printf("\n>>Enter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            FCFS();
            break;
        case 2:
            SCAN();
            break;
        case 3:
            C_SCAN();
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice\n");
        }
    }
}
```

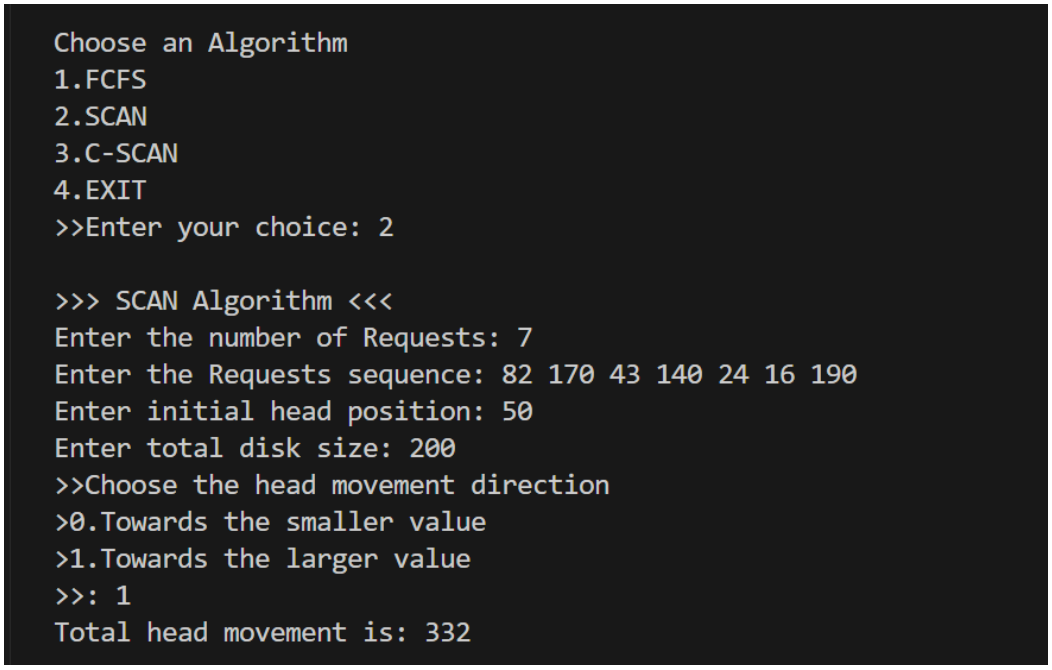
<u>OUTPUT:</u>

a) FCFS

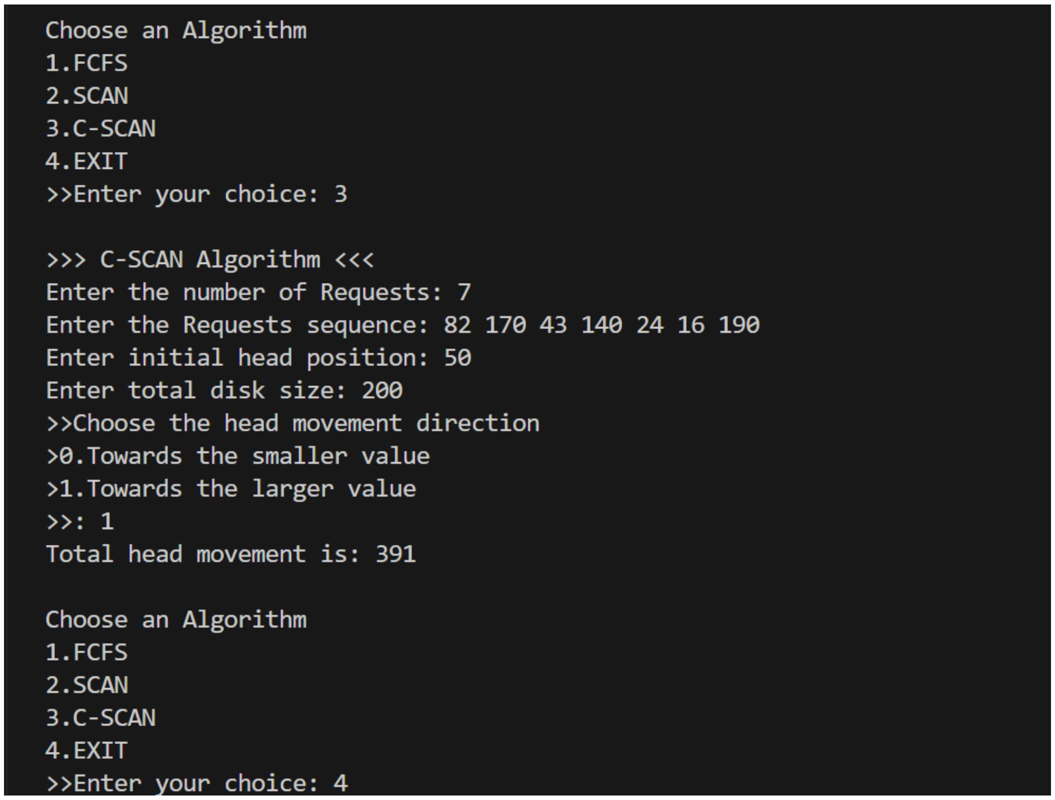


```
Disk Scheduling Algorithms

Choose an Algorithm
1.FCFS
2.SCAN
3.C-SCAN
4.EXIT
>>Enter your choice: 1

>>> FCFS Algorithm <<<
Enter the number of Requests: 7
Enter the Requests sequence: 82 170 43 140 24 16 190
Enter initial head position: 50
Total head moment is: 642
```

b) SCAN


```
Choose an Algorithm
1.FCFS
2.SCAN
3.C-SCAN
4.EXIT
>>Enter your choice: 2

>>> SCAN Algorithm <<<
Enter the number of Requests: 7
Enter the Requests sequence: 82 170 43 140 24 16 190
Enter initial head position: 50
Enter total disk size: 200
>>Choose the head movement direction
>0.Towards the smaller value
>1.Towards the larger value
>>: 1
Total head movement is: 332
```

c) C-SCAN


```
Choose an Algorithm
1.FCFS
2.SCAN
3.C-SCAN
4.EXIT
>>Enter your choice: 3

>>> C-SCAN Algorithm <<<
Enter the number of Requests: 7
Enter the Requests sequence: 82 170 43 140 24 16 190
Enter initial head position: 50
Enter total disk size: 200
>>Choose the head movement direction
>0.Towards the smaller value
>1.Towards the larger value
>>: 1
Total head movement is: 391

Choose an Algorithm
1.FCFS
2.SCAN
3.C-SCAN
4.EXIT
>>Enter your choice: 4
```

PROGRAM-15

Write a C program to simulate disk scheduling algorithms

a) SSTF

b) LOOK

c) c-LOOK

```
#include <stdio.h>

#include <stdlib.h>


void SSTF()

{

    int RQ[100], i, n, TotalHeadMoment = 0, initial, count = 0;

    printf("\n>>> SSTF Algorithm <<<");

    printf("\nEnter the number of Requests: ");

    scanf("%d", &n);

    printf("Enter the Requests sequence: ");

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

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

    printf("Enter initial head position: ");

    scanf("%d", &initial);

    while (count != n)

    {

        int min = 1000, d, index;

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

        {

            d = abs(RQ[i] - initial);

            if (min > d)

            {
                min = d;

                index = i;

            }

        }

        TotalHeadMoment = TotalHeadMoment + min;

        initial = RQ[index];

        RQ[index] = 1000;

        count++;

    }


    printf("Total head movement is: %d", TotalHeadMoment);

}


void LOOK()

{

    int RQ[100], i, j, n, TotalHeadMoment = 0, initial, size, move;

    printf("\n>>> LOOK Algorithm <<<");

    printf("\nEnter the number of Requests: ");

    scanf("%d", &n);

    printf("Enter the Requests sequence: ");

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

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

    printf("Enter initial head position: ");

    scanf("%d", &initial);

    printf(">>Choose the head movement direction\n>0.Towards the smaller value\n>1.Towards 
the larger value\n>>: ");

    scanf("%d", &move);


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

```
{
    for (j = 0; j < n - i - 1; j++)
    {
        if (RQ[j] > RQ[j + 1])
        {
            int temp;
            temp = RQ[j];
            RQ[j] = RQ[j + 1];
            RQ[j + 1] = temp;
        }
    }
}

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

if (move == 1)
{
    for (i = index; i < n; i++)
    {
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
```

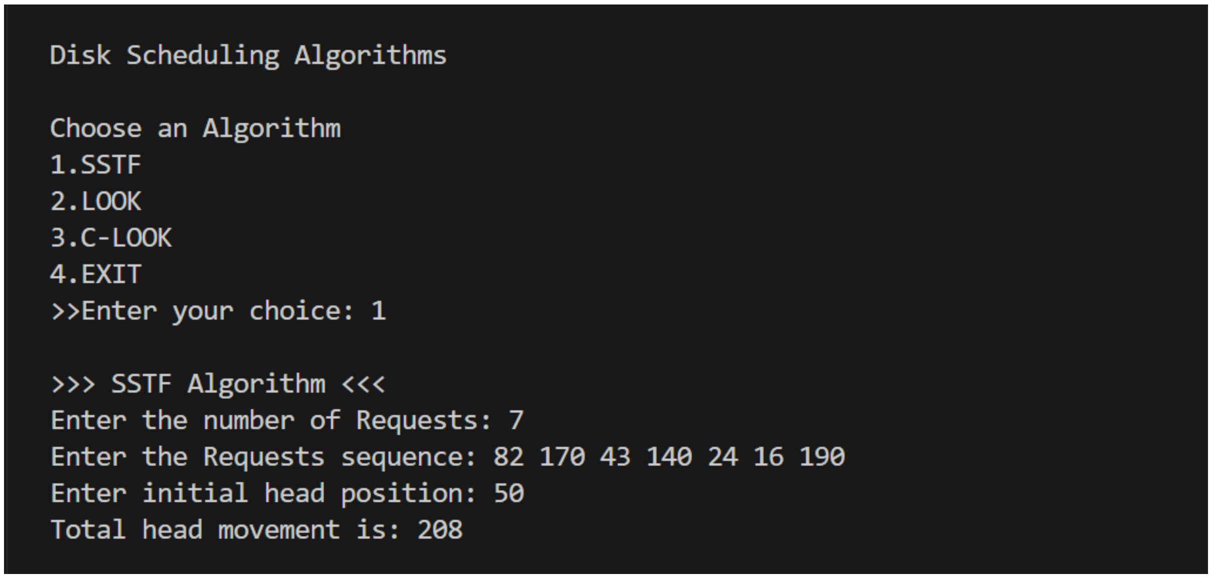
```
            initial = RQ[i];
        }

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

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

    printf("Total head movement is: %d", TotalHeadMoment);
}

void C_LOOK()
```

```
{
   int RQ[100], i, j, n, TotalHeadMoment = 0, initial, size, move;


   printf("\n>>> C-LOOK Algorithm <<<");
   printf("\nEnter the number of Requests: ");
   scanf("%d", &n);
   printf("Enter the Requests sequence: ");
   for (i = 0; i < n; i++)
      scanf("%d", &RQ[i]);
   printf("Enter initial head position: ");
   scanf("%d", &initial);


   printf(">>Choose the head movement direction\n>0.Towards the smaller value\n>1.Towards
the larger value\n>>: ");
   scanf("%d", &move);


   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n - i - 1; j++)
      {
         if (RQ[j] > RQ[j + 1])
         {
            int temp;
            temp = RQ[j];
            RQ[j] = RQ[j + 1];
            RQ[j + 1] = temp;
         }
      }
   }
```

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

if (move == 1)
{
    for (i = index; i < n; i++)
    {
        TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial);
        initial = RQ[i];
    }

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

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

    printf("Total head movement is: %d", TotalHeadMoment);
}

void main()
{
    int ch;
    printf("\nDisk Scheduling Algorithms");
    while (1)
    {
        printf("\n\nChoose an Algorithm\n");
        printf("1.SSTF\n2.LOOK\n3.C-LOOK\n4.EXIT");
        printf("\n>>Enter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            SSTF();
            break;
        case 2:
            LOOK();
            break;
        case 3:
            C_LOOK();
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice\n");
        }
    }
}
```

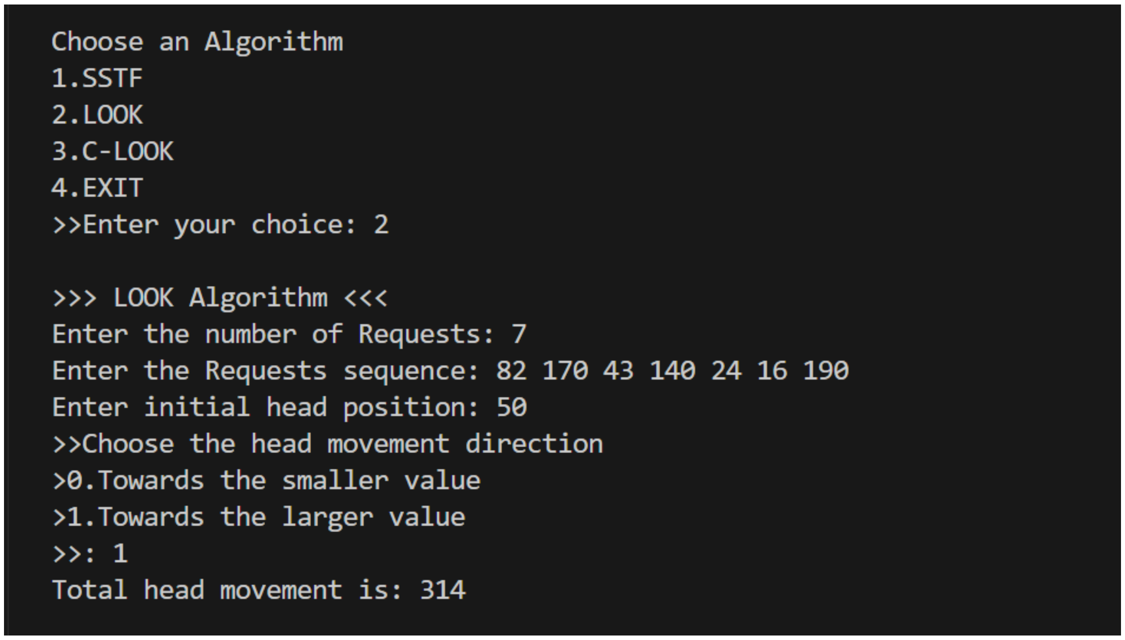
OUTPUT:

a) SSTF



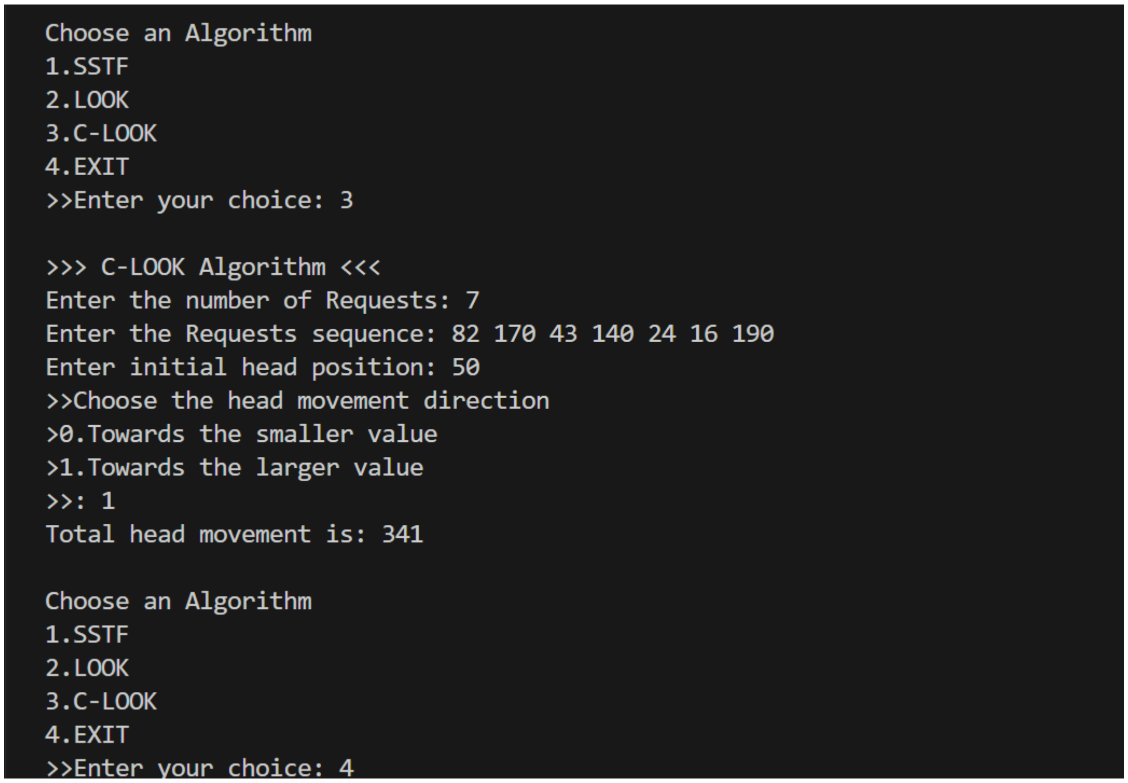
```
Disk Scheduling Algorithms

Choose an Algorithm
1.SSTF
2.LOOK
3.C-LOOK
4.EXIT
>>Enter your choice: 1

>>> SSTF Algorithm <<<
Enter the number of Requests: 7
Enter the Requests sequence: 82 170 43 140 24 16 190
Enter initial head position: 50
Total head movement is: 208
```

b) LOOK



```
Choose an Algorithm
1.SSTF
2.LOOK
3.C-LOOK
4.EXIT
>>Enter your choice: 2

>>> LOOK Algorithm <<<
Enter the number of Requests: 7
Enter the Requests sequence: 82 170 43 140 24 16 190
Enter initial head position: 50
>>Choose the head movement direction
>0.Towards the smaller value
>1.Towards the larger value
>>: 1
Total head movement is: 314
```

c) c-LOOK



```
Choose an Algorithm
1.SSTF
2.LOOK
3.C-LOOK
4.EXIT
>>Enter your choice: 3

>>> C-LOOK Algorithm <<<
Enter the number of Requests: 7
Enter the Requests sequence: 82 170 43 140 24 16 190
Enter initial head position: 50
>>Choose the head movement direction
>0.Towards the smaller value
>1.Towards the larger value
>>: 1
Total head movement is: 341

Choose an Algorithm
1.SSTF
2.LOOK
3.C-LOOK
4.EXIT
>>Enter your choice: 4
```