VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

Operating Systems

(22CS4PCOPS)

Submitted by:

Vismitha Raj S Doshi (1WA23CS047)

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

Feb 2025 - June 2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

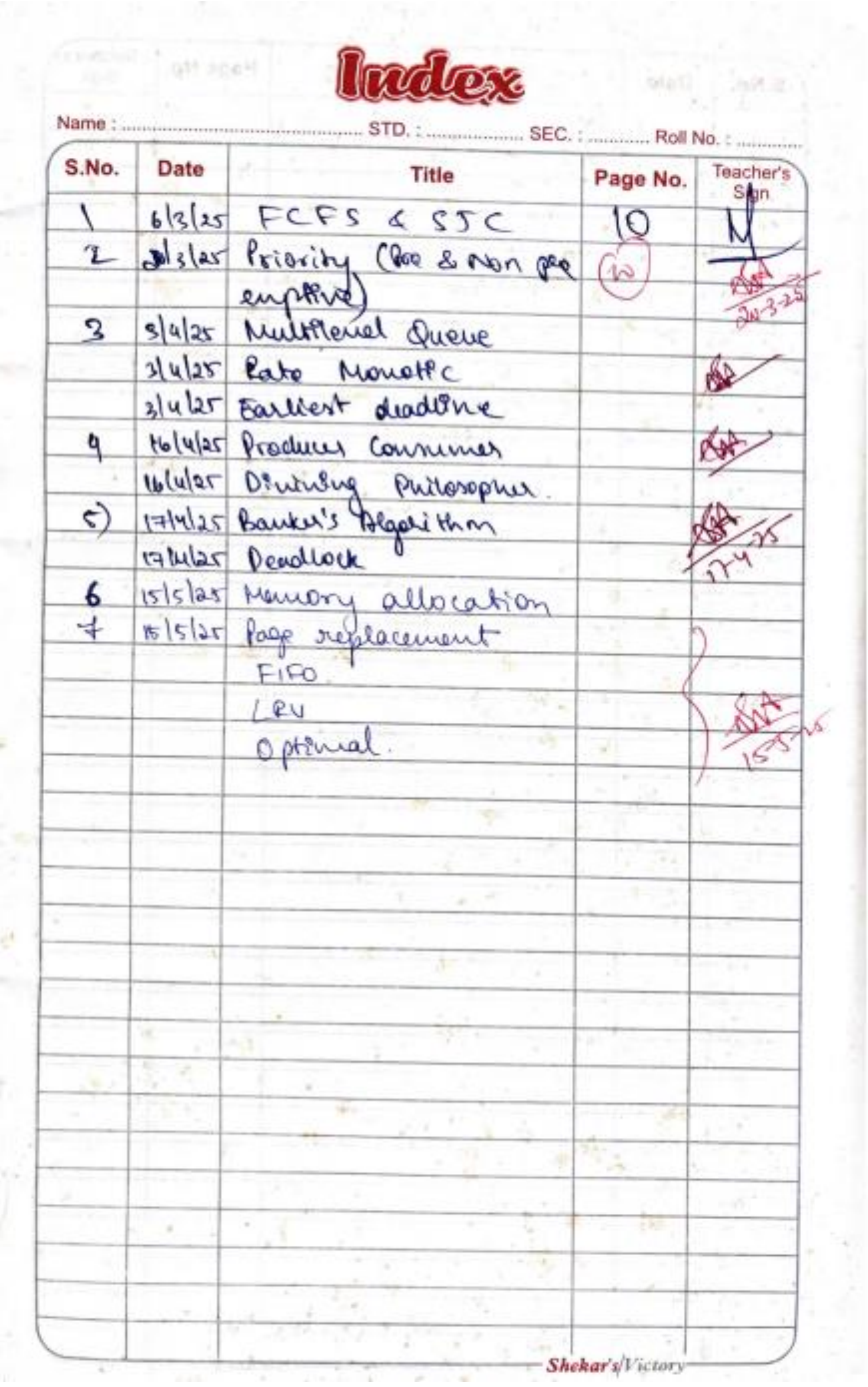
CERTIFICATE

This is to certify that the Lab work entitled "**Operating Systems**" carried out by **Vismitha Raj S Doshi(1WA23CS047),** 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 year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of **Operating Systems - (22CS4PCOPS)** work prescribed for the said degree.

Dr Seema Patil
Associate Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Table Of Contents



Index

Name : .... STD. : .... SEC. : .... Roll No. : ....

Course Outcomes

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.

GITHUB LINK:

https://github.com/vismitharaj/OS-lab

Experiments

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

(a) FCFS

(b) SJF

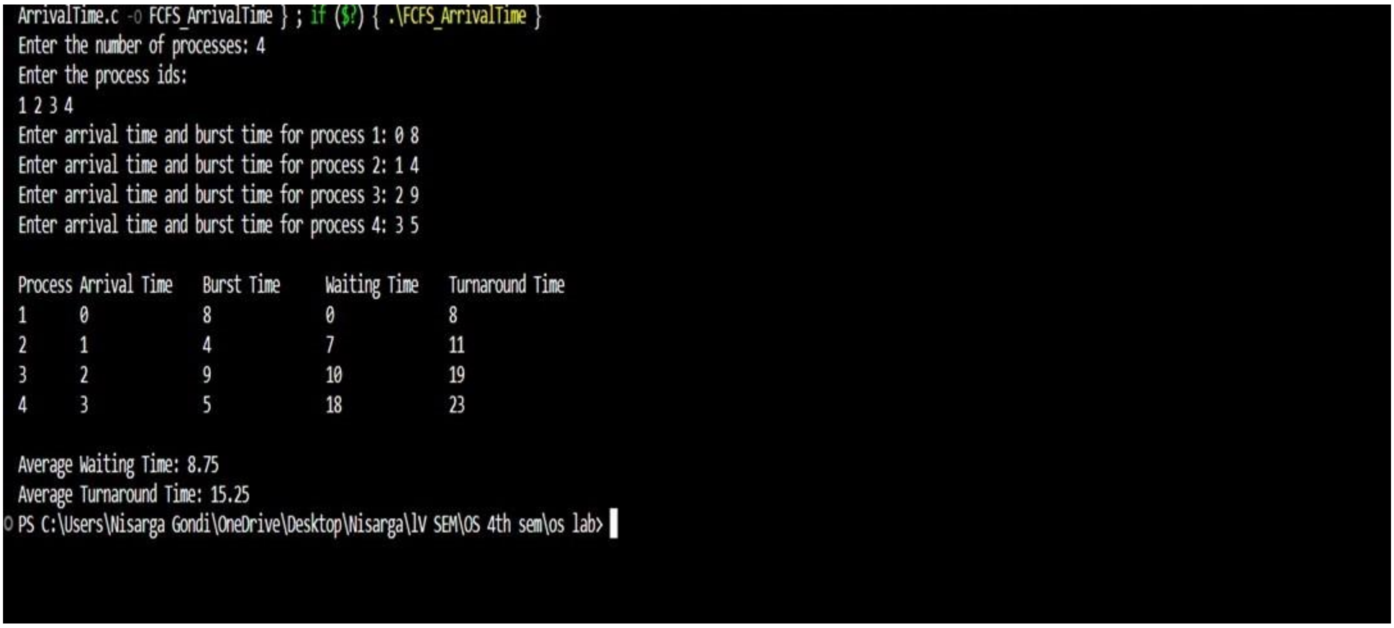
```
#include<stdio.h>
   int n, i, j, pos, temp, choice, Burst_time[20], Waiting_time[20],
   Turn_around_time[20], process[20], total=0;
   float avg_Turn_around_time=0, avg_Waiting_time=0;

int FCFS()
{
   Waiting_time[0]=0;

   for(i=1;i<n;i++)
   {
      Waiting_time[i]=0;
      for(j=0;j<i;j++)
         Waiting_time[i]+=Burst_time[j];
   }

   printf("\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurnaround Time");

   for(i=0;i<n;i++)
   {
      Turn_around_time[i]=Burst_time[i]+Waiting_time[i];
      avg_Waiting_time+=Waiting_time[i];
      avg_Turn_around_time+=Turn_around_time[i];

printf("\nP[%d]\t\t%d\t\t\t%d\t\t\t\t%d",i+1,Burst_time[i],Waiting_time[i],Turn_around_t
ime[i]);
   }

   avg_Waiting_time =(float)(avg_Waiting_time)/(float)i;
   avg_Turn_around_time=(float)(avg_Turn_around_time)/(float)i;
   printf("\nAverage Waiting Time:%.2f",avg_Waiting_time);
   printf("\nAverage Turnaround Time:%.2f\n",avg_Turn_around_time);

   return 0;
}
```

```
int SJF()
{
   //sorting
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(Burst_time[j]<Burst_time[pos])
            pos=j;
      }

      temp=Burst_time[i];
      Burst_time[i]=Burst_time[pos];
      Burst_time[pos]=temp;

      temp=process[i];
      process[i]=process[pos];
      process[pos]=temp;
   }
      Waiting_time[0]=0;


   for(i=1;i<n;i++)
   {
      Waiting_time[i]=0;

      for(j=0;j<i;j++)
         Waiting_time[i]+=Burst_time[j];

      total+=Waiting_time[i];
   }

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

   printf("\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurnaround Time");

   for(i=0;i<n;i++)
   {
      Turn_around_time[i]=Burst_time[i]+Waiting_time[i];
      total+=Turn_around_time[i];

printf("\nP[%d]\t\t%d\t\t\t%d\t\t\t\t%d",process[i],Burst_time[i],Waiting_time[i],Turn_ar
ound_time[i]);
```

```
    }

    avg_Turn_around_time=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_Waiting_time);
    printf("\nAverage Turnaround Time=%f\n",avg_Turn_around_time);
}

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

    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d]:",i+1);
        scanf("%d",&Burst_time[i]);
        process[i]=i+1;
    }

    while(1)
    {   printf("\n-----MAIN MENU -----\n");
        printf("1. FCFS Scheduling\n2. SJF Scheduling\n");
        printf("\nEnter your choice:");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: FCFS();
            break;

            case 2: SJF();
            break;

            default: printf("Invalid Input!!!");
        }
    }
    return 0;
}
```

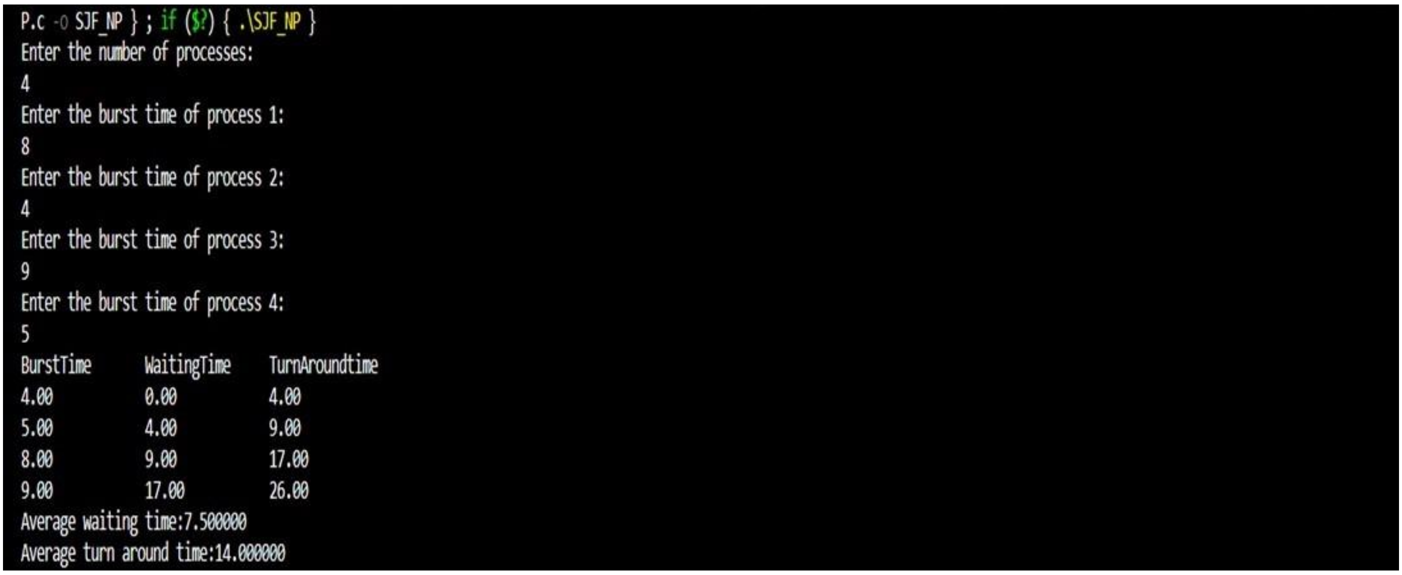
Output:

a.

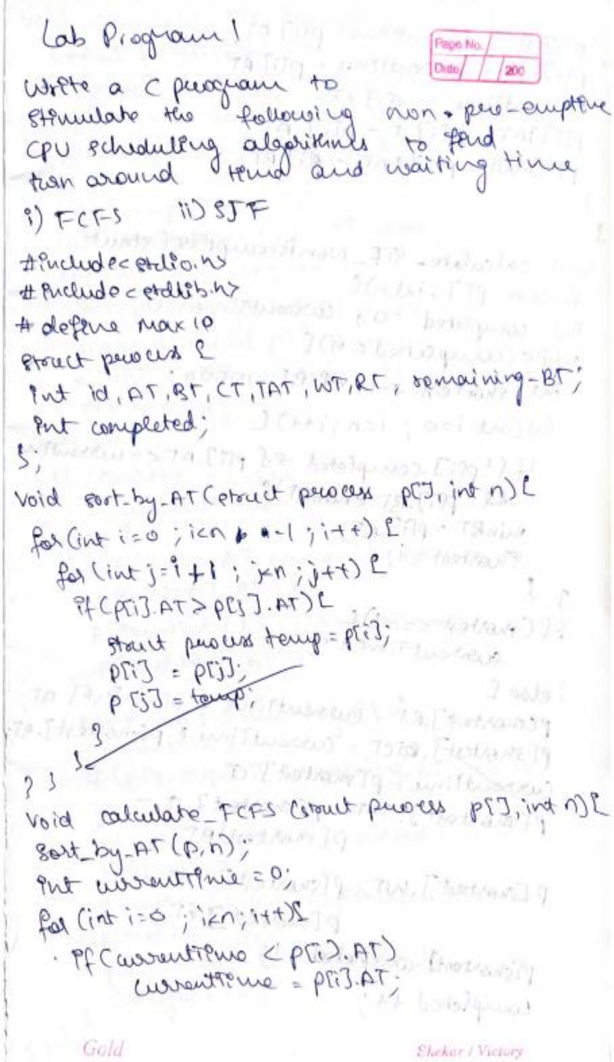


```
ArrivalTime.c -o FCFS_ArrivalTime } ; if ($?) { .\FCFS_ArrivalTime }
Enter the number of processes: 4
Enter the process ids:
1 2 3 4
Enter arrival time and burst time for process 1: 0 8
Enter arrival time and burst time for process 2: 1 4
Enter arrival time and burst time for process 3: 2 9
Enter arrival time and burst time for process 4: 3 5

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
1       0               8               0               8
2       1               4               7               11
3       2               9               10              19
4       3               5               18              23

Average Waiting Time: 8.75
Average Turnaround Time: 15.25
○ PS C:\Users\Nisarga Gondi\OneDrive\Desktop\Nisarga\lV SEM\OS 4th sem\os lab>
```

b.



```
P.c -o SJF_NP } ; if ($?) { .\SJF_NP }
Enter the number of processes:
4
Enter the burst time of process 1:
8
Enter the burst time of process 2:
4
Enter the burst time of process 3:
9
Enter the burst time of process 4:
5
BurstTime       WaitingTime     TurnAroundtime
4.00            0.00            4.00
5.00            4.00            9.00
8.00            9.00            17.00
9.00            17.00           26.00
Average waiting time:7.500000
Average turn around time:14.000000
```



Lab Program 1

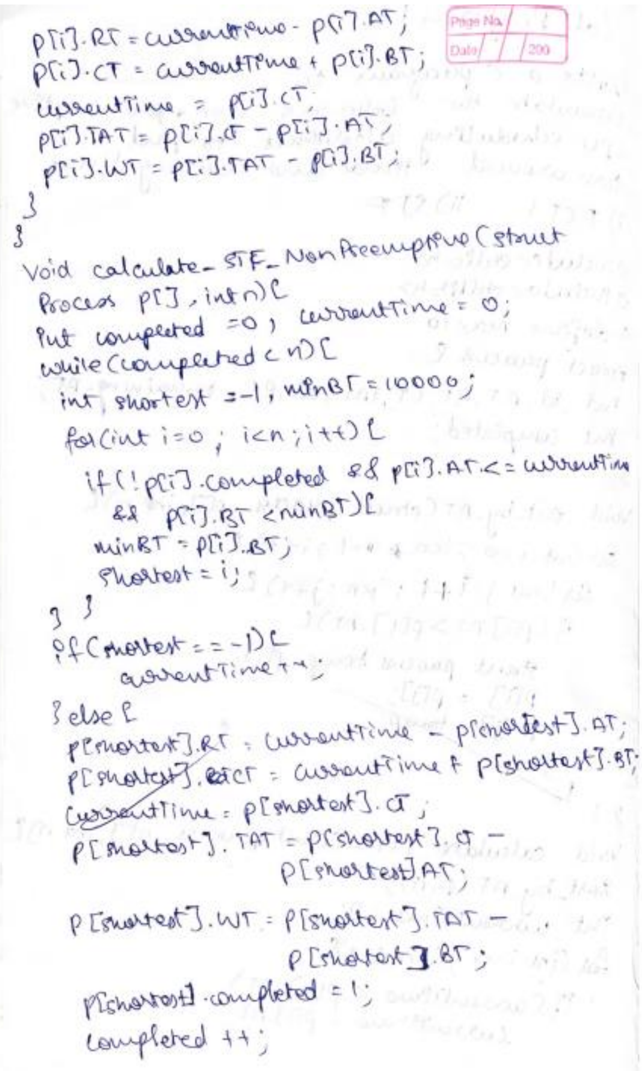
Page No. / Date / / 200

Write a C program to stimulate the following non-pre-emptive CPU scheduling algorithms to find turn around time and waiting time

i) FCFS   ii) SJF

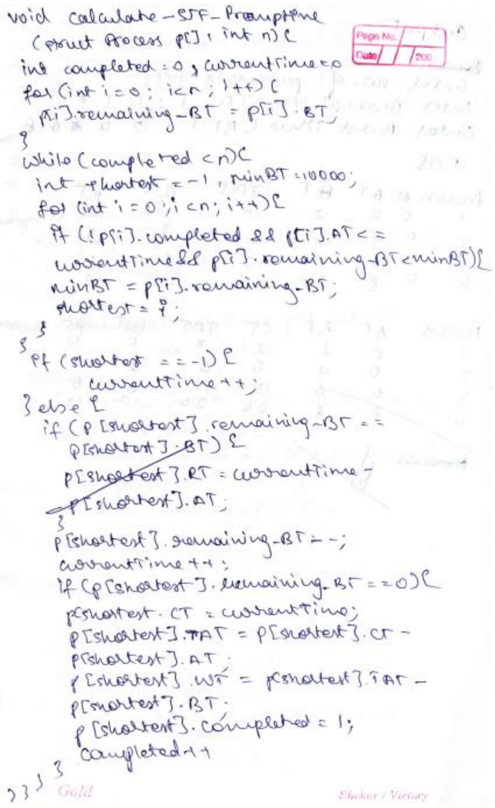
```
#include<stdio.h>
#include<stdlib.h>
#define max 10
struct process {
 int id, AT, BT, CT, TAT, WT, RT, remaining_BT;
 int completed;
};
void sort_by_AT(struct process p[], int n){
 for(int i=0 ; i<n-1 ; i++) {
  for(int j=i+1 ; j<n ; j++) {
   if(p[i].AT > p[j].AT){
     struct process temp = p[i];
     p[i] = p[j];
     p[j] = temp;
   }
  }
 }}
void calculate_FCFS (struct process p[], int n){
 sort_by_AT(p, n);
 int currentTime = 0;
 for (int i=0 ; i<n; i++){
  if(currentTime < p[i].AT)
       currentTime = p[i].AT;
```

Gold   Shekar / Victory



Page No. / Date / / 200

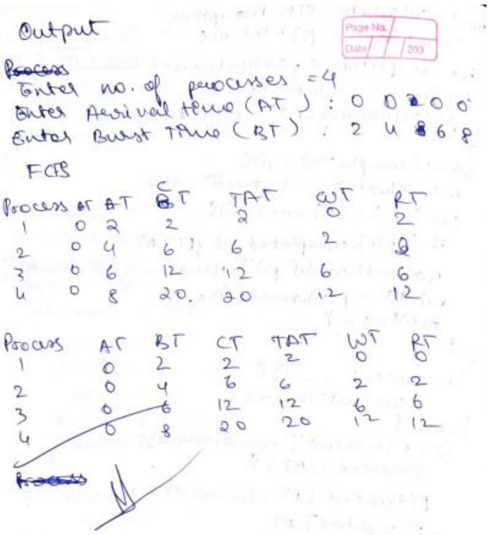
```
p[i].RT = currentTime - p[i].AT;
p[i].CT = currentTime + p[i].BT;
currentTime = p[i].CT;
p[i].TAT = p[i].CT - p[i].AT;
p[i].WT = p[i].TAT - p[i].BT;
}
}
void calculate_SJF_NonPreemptive(struct
 Process p[], int n){
 int completed =0, currentTime = 0;
 while(completed < n){
  int shortest =-1, minBT = 10000;
  for(int i=0 ; i<n; i++){
   if(!p[i].completed && p[i].AT <= currentTime
     && p[i].BT < minBT){
   minBT = p[i].BT;
   shortest = i;
  }}
  if(shortest == -1){
     currentTime++;
  }else{
   p[shortest].RT = currentTime - p[shortest].AT;
   p[shortest].~~BT~~CT = currentTime + p[shortest].BT;
   currentTime = p[shortest].CT;
   p[shortest].TAT = p[shortest].CT -
                 p[shortest].AT;

   p[shortest].WT = p[shortest].TAT -
                 p[shortest].BT;

   p[shortest].completed = 1;
   completed ++;
```



Page No. / 1 / Date / / 200

```
void calculate_SJF_Preemptive
 (struct Process p[], int n){
 int completed = 0, currentTime = 0
 for (int i=0; i<n; i++){
  p[i].remaining_BT = p[i].BT;
 }
 while (completed < n){
  int shortest = -1, minBT =10000;
  for (int i = 0; i<n; i++){
   if (!p[i].completed && p[i].AT <=
   currentTime && p[i].remaining_BT<minBT){
   minBT = p[i].remaining_BT;
   shortest = i;
  }
 }
  if (shortest == -1){
      currentTime++;
 }else{
  if (p[shortest].remaining_BT ==
    p[shortest].BT){
   p[shortest].RT = currentTime -
   p[shortest].AT;
  }
  p[shortest].remaining_BT--;
  currentTime++;
  if (p[shortest].remaining_BT == 0){
   p[shortest.CT = currentTime;
   p[shortest].TAT = p[shortest].CT -
   p[shortest].AT;
   p[shortest].WT = p[shortest].TAT -
   p[shortest].BT;
   p[shortest].completed = 1;
   completed++
 }}
}}
```

Gold   Shekar / Victory



Output

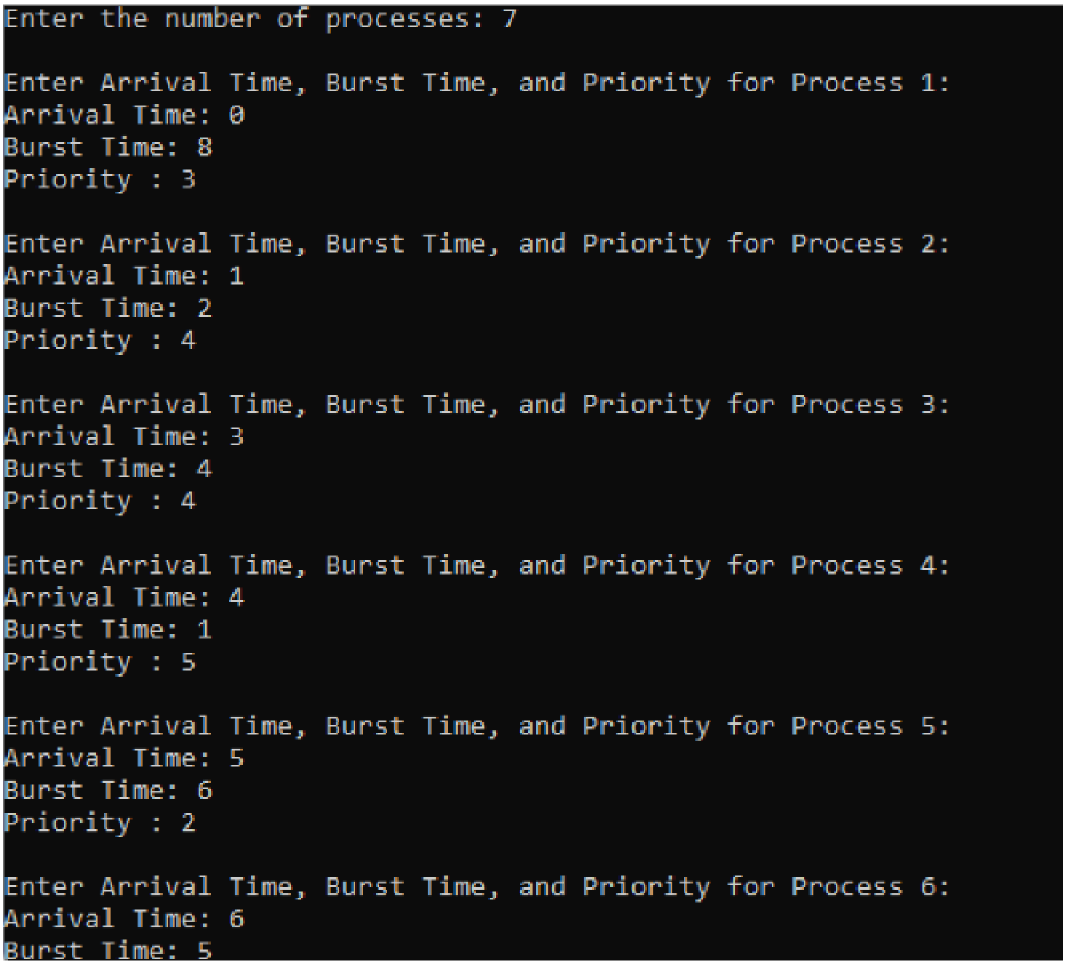
Page No. / Date / / 200

```
~~Process~~
Enter no. of processes =4
Enter Arrival time (AT) : 0 0 0 0
Enter Burst Time (BT) : 2 4 6 8
 FCFS
Process AT AT  CT   TAT   WT   RT
  1     0  2   2     2    0    2
  2     0  4   6     6    2    2
  3     0  6   12    12   6    6
  4     0  8   20.   20   12   12

Process   AT  BT   CT   TAT   WT   RT
  1       0   2    2     2    0    0
  2       0   4    6     6    2    2
  3       0   6    12    12   6    6
  4       0   8    20    20   12   12
~~Process~~
```

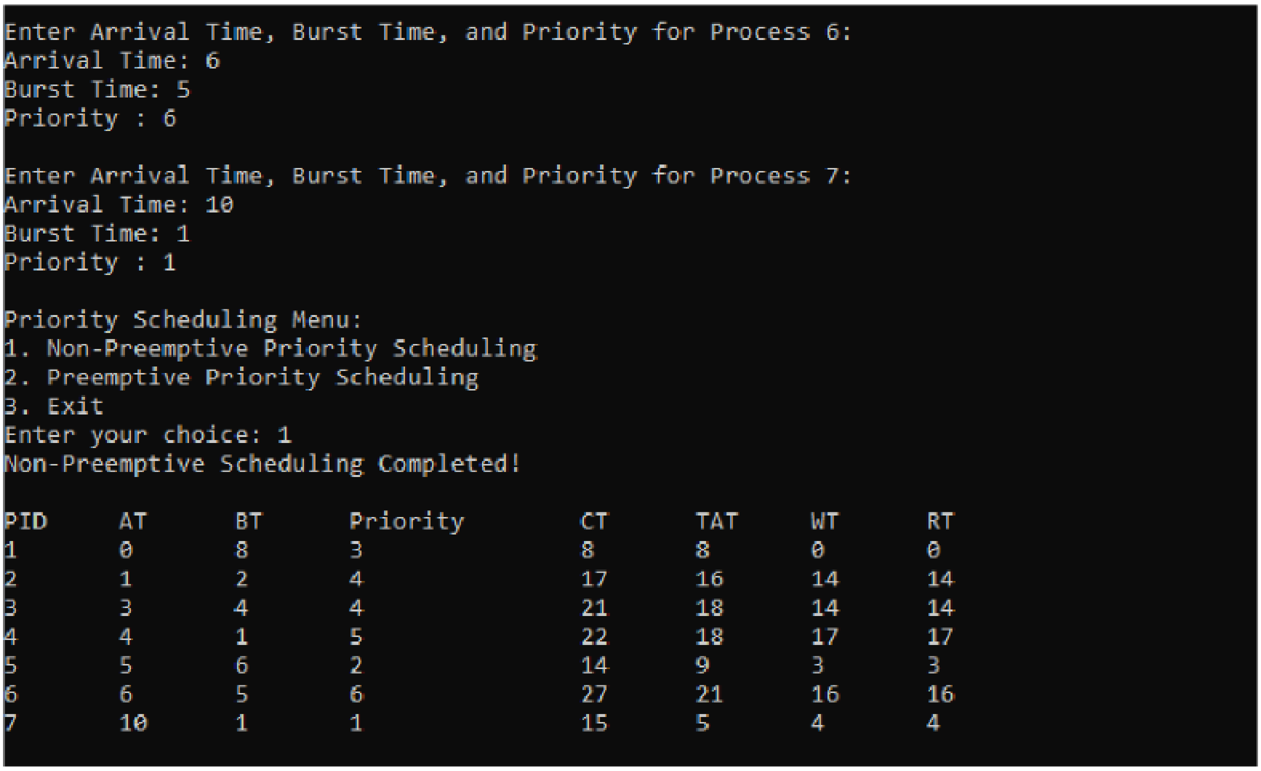
2.Priority

```
#include <stdio.h>
#define MAX 10
typedef struct {
    int pid, at, bt, pt, remaining_bt, ct, tat, wt, rt, is_completed, st;
} Process;
void nonPreemptivePriority(Process p[], int n) {
    int time = 0, completed = 0;
    while (completed < n) {
        int highest_priority = 9999, selected = -1;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && !p[i].is_completed && p[i].pt < highest_priority) {
                highest_priority = p[i].pt;
                selected = i;
            }
        }
        if (selected == -1) {
            time++;
            continue;
        }
        if (p[selected].rt == -1) {
            p[selected].st = time;  // Start time
            p[selected].rt = time - p[selected].at;
        }
        time += p[selected].bt;
        p[selected].ct = time;
        p[selected].tat = p[selected].ct - p[selected].at;
        p[selected].wt = p[selected].tat - p[selected].bt;
        p[selected].is_completed = 1;
        completed++;
    }
}
void preemptivePriority(Process p[], int n) {
    int time = 0, completed = 0;

    while (completed < n) {
        int highest_priority = 9999, selected = -1;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].remaining_bt > 0 && p[i].pt < highest_priority) {
                highest_priority = p[i].pt;
                selected = i;
            }
        }
        if (selected == -1) {
            time++;
            continue;
        }
        if (p[selected].rt == -1) {
            p[selected].st = time;  // Start time
            p[selected].rt = time - p[selected].at;
        }
        p[selected].remaining_bt--;
        time++;

        if (p[selected].remaining_bt == 0) {
            p[selected].ct = time;
            p[selected].tat = p[selected].ct - p[selected].at;
            p[selected].wt = p[selected].tat - p[selected].bt;
            completed++;
        }
    }
}
void displayProcesses(Process p[], int n) {
    float avg_tat = 0, avg_wt = 0, avg_rt = 0;

    printf("\nPID\tAT\tBT\tPriority\tCT\tTAT\tWT\tRT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t\t%d\t%d\t%d\t%d\n",
               p[i].pid, p[i].at, p[i].bt, p[i].pt, p[i].ct, p[i].tat, p[i].wt, p[i].rt);
        avg_tat += p[i].tat;
        avg_wt += p[i].wt;
        avg_rt += p[i].rt;
    }

    printf("\nAverage TAT: %.2f", avg_tat / n);
    printf("\nAverage WT: %.2f", avg_wt / n);
    printf("\nAverage RT: %.2f\n", avg_rt / n);
}
int main() {
    Process p[MAX];
    int n, choice;

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

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("\nEnter Arrival Time, Burst Time, and Priority for Process %d:\n", p[i].pid);
        printf("Arrival Time: ");
        scanf("%d", &p[i].at);
        printf("Burst Time: ");
        scanf("%d", &p[i].bt);
        printf("Priority : ");
        scanf("%d", &p[i].pt);
        p[i].remaining_bt = p[i].bt;
        p[i].is_completed = 0;
        p[i].rt = -1;
    }

    while (1) {
        printf("\nPriority Scheduling Menu:\n");
        printf("1. Non-Preemptive Priority Scheduling\n");
        printf("2. Preemptive Priority Scheduling\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                nonPreemptivePriority(p, n);
                printf("Non-Preemptive Scheduling Completed!\n");
                displayProcesses(p, n);
                break;
            case 2:
                preemptivePriority(p, n);
                printf("Preemptive Scheduling Completed!\n");
                displayProcesses(p, n);
                break;
            case 3:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }


    return 0;
}
```

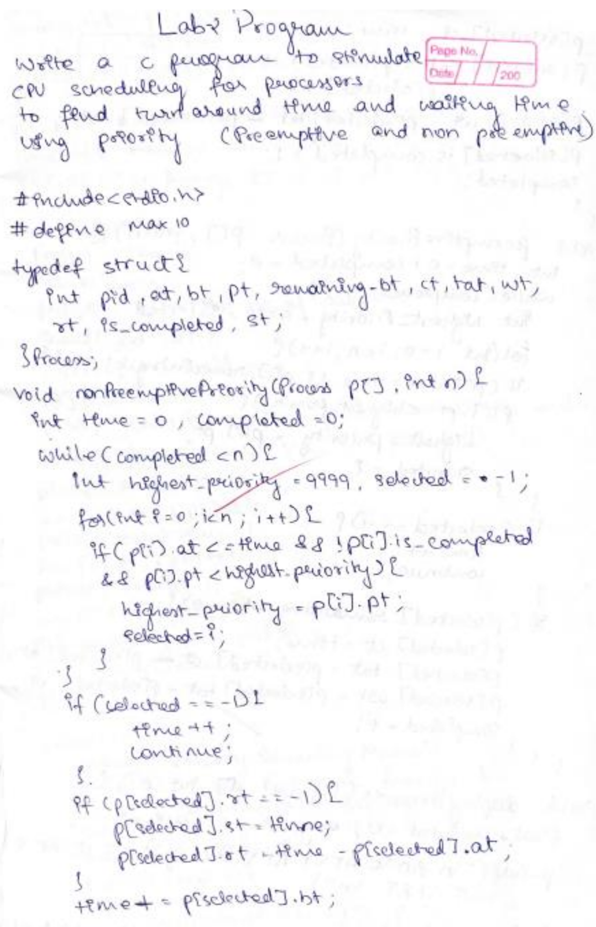
Output:



```
Enter the number of processes: 7

Enter Arrival Time, Burst Time, and Priority for Process 1:
Arrival Time: 0
Burst Time: 8
Priority : 3

Enter Arrival Time, Burst Time, and Priority for Process 2:
Arrival Time: 1
Burst Time: 2
Priority : 4

Enter Arrival Time, Burst Time, and Priority for Process 3:
Arrival Time: 3
Burst Time: 4
Priority : 4

Enter Arrival Time, Burst Time, and Priority for Process 4:
Arrival Time: 4
Burst Time: 1
Priority : 5

Enter Arrival Time, Burst Time, and Priority for Process 5:
Arrival Time: 5
Burst Time: 6
Priority : 2

Enter Arrival Time, Burst Time, and Priority for Process 6:
Arrival Time: 6
Burst Time: 5
```



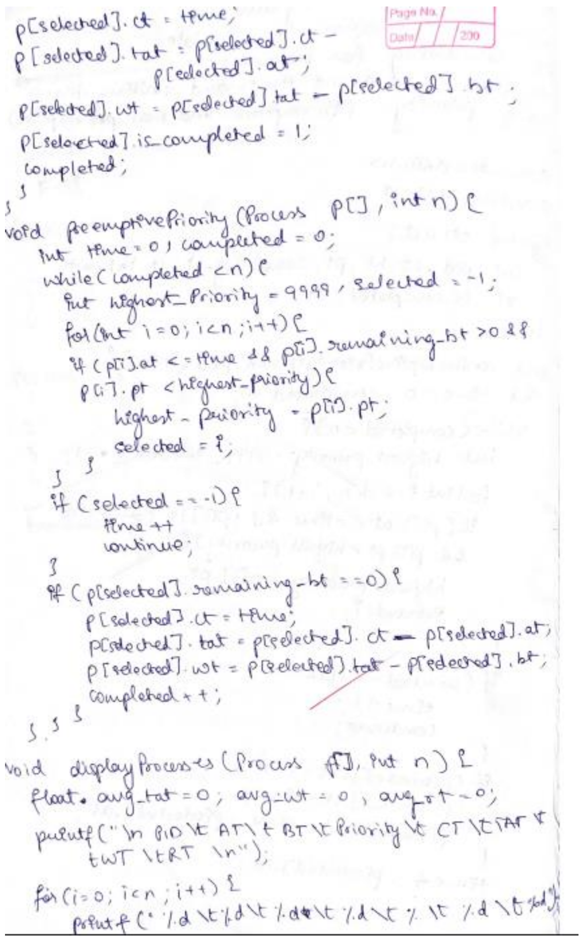
```
Enter Arrival Time, Burst Time, and Priority for Process 6:
Arrival Time: 6
Burst Time: 5
Priority : 6

Enter Arrival Time, Burst Time, and Priority for Process 7:
Arrival Time: 10
Burst Time: 1
Priority : 1

Priority Scheduling Menu:
1. Non-Preemptive Priority Scheduling
2. Preemptive Priority Scheduling
3. Exit
Enter your choice: 1
Non-Preemptive Scheduling Completed!

PID     AT      BT      Priority        CT      TAT     WT      RT
1       0       8       3               8       8       0       0
2       1       2       4               17      16      14      14
3       3       4       4               21      18      14      14
4       4       1       5               22      18      17      17
5       5       6       2               14      9       3       3
6       6       5       6               27      21      16      16
7       10      1       1               15      5       4       4
```



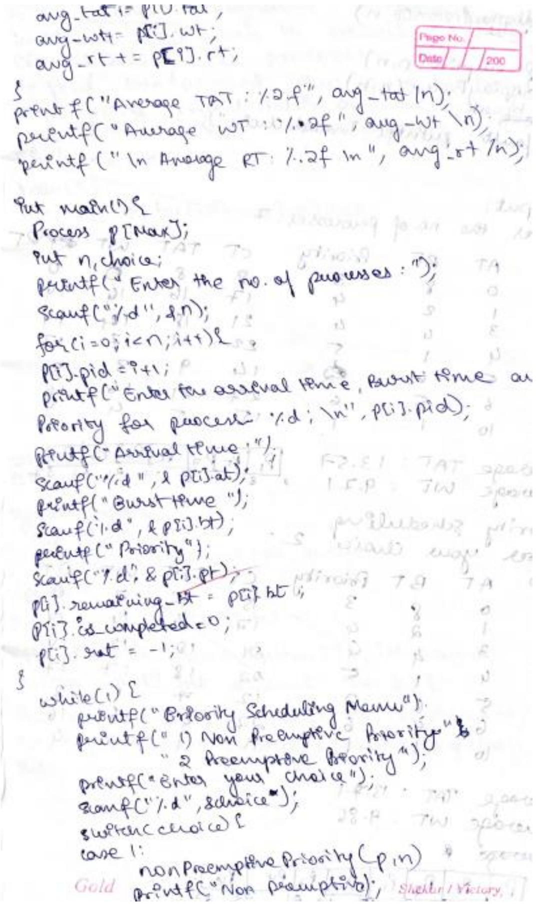
Lab2 Program

Write a c program to stimulate CPU scheduling for processors to find turnaround time and waiting time using priority (Preemptive and non preemptive)

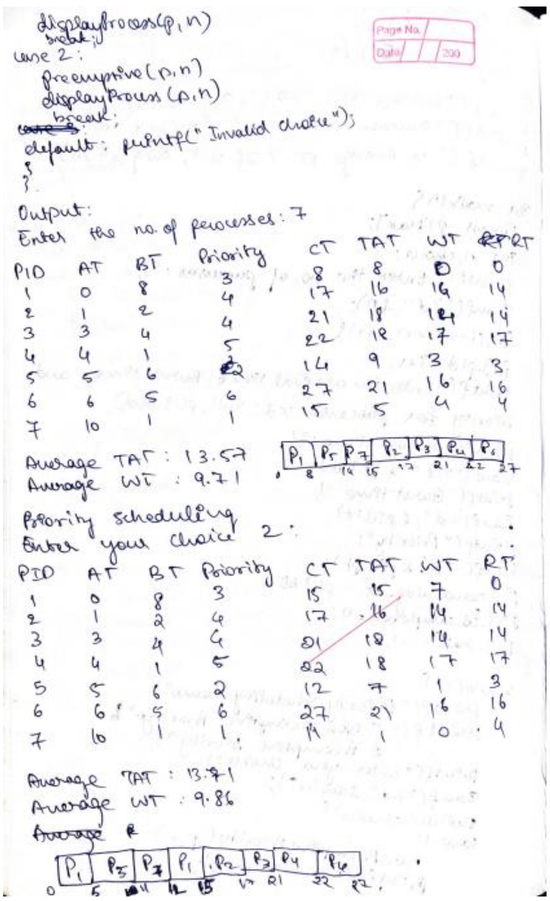
```
#include<stdio.h>
#define max 10

typedef struct{
  int pid, at, bt, pt, remaining-bt, ct, tat, wt,
   rt, is_completed, st;
}Process;
void nonPreemptivePriority(Process p[], int n) {
 int time = 0, completed =0;
 while(completed <n){
   int highest_priority = 9999, selected = -1;
    for(int i=0 ; i<n ; i++){
     if(p[i].at <= time && !p[i].is_completed
      && p[i].pt <highest_priority){
        highest_priority = p[i].pt;
        selected=i;
     }
   }
  if (selected ==-1){
       time++;
       continue;
   }.
  if (p[selected].rt ==-1){
      p[selected].st = time;
      p[selected].rt = time - p[selected].at;
  }
  time+= p[selected].bt;
```



```
p[selected].ct = time;
p[selected].tat = p[selected].ct -
            p[selected].at;
p[selected].wt = p[selected] tat - p[selected] bt;
p[selected].is_completed = 1;
completed;
}
}
void preemptivePriority(Process p[], int n) {
  int time=0, completed = 0;
  while(completed <n){
   int highest_Priority = 9999, selected = -1;
   for(int i=0; i<n ;i++){
   if (p[i].at <= time && p[i].remaining-bt >0 &&
    p[i].pt <highest_priority){
       highest - priority = p[i].pt;
       selected = i;
     }
  }
  if (selected == -1){
      time ++
      continue;
  }
  if (p[selected].remaining-bt ==0){
     p[selected].ct = time;
     p[selected].tat = p[selected].ct - p[selected].at;
     p[selected].wt = p[selected].tat - p[selected].bt;
     completed++;
} } }

void displayProcesses(Process p[], int n) {
 float avg_tat =0; avg_wt = 0, avg_rt =0;
 printf("\n PID \t AT\t BT\t Priority \t CT \t TAT \t
      tWT \tRT \n");
 for(i=0; i<n ; i++) {
   printf(" %d \t %d\t %d\t %d\t % \t %d \t %d");
```



```
 avg_tat += p[i].tat;
 avg-wt+= p[i].wt;
 avg_rt += p[i].rt;
}
printf("Average TAT : %.2f", avg_tat /n);
printf("Average WT : %.2f", avg-wt \n);
printf(" \n Average RT : %.2f \n", avg_rt /n);

int main(){
 Process p[Max];
 int n, choice;
 printf(" Enter the no. of processes : ");
 scanf("%d", &n);
 for(i=0; i<n; i++){
  p[i].pid =i+1;
  printf("Enter the arrival time, Burst time and
 Priority for process %d; \n", p[i].pid);
 printf("Arrival time :");
 scanf("%d", & p[i].at);
 printf("Burst time ");
 scanf("%d", & p[i].bt);
 printf(" Priority");
 scanf("%d", & p[i].pt);
 p[i].remaining_bt = p[i] bt;
 p[i].is_completed=0;
 p[i].rat = -1;
}
 while(1) {
   printf("Priority Scheduling Menu");
   printf(" 1) Non Preemptive Priority"
           " 2 Preemptive Priority");
   printf("Enter your choice");
   scanf("%d", &choice);
   switch(choice) {
   case 1:
         nonPreemptivePriority(p,n)
         printf("Non preemptive");
```



```
 displayProcess(p,n)
 break;
case 2:
 preemptive(p,n)
 displayProcess(p,n)
 break;
default: printf(" Invalid choice");
}
}.
```

Output:

Enter the no. of processes: 7

PID	AT	BT	Priority	CT	TAT	WT	~~RT~~RT
1	0	8	3	8	8	0	0
2	1	2	4	17	16	16	14
3	3	4	4	21	18	14	14
4	4	1	5	22	18	17	17
5	5	6	2	14	9	3	3
6	6	5	6	27	21	16	16
7	10	1	1	15	5	4	4

Average TAT : 13.57
Average WT : 9.71

```
| P1 | P5 | P7 | P2 | P3 | P4 | P6 |
0    8    14   15   17   21   22   27
```

Priority scheduling
Enter your choice 2 :

PID	AT	BT	Priority	CT	TAT	WT	RT
1	0	8	3	15	15	7	0
2	1	2	4	17	16	14	14
3	3	4	4	21	18	14	14
4	4	1	5	22	18	17	17
5	5	6	2	12	7	1	3
6	6	5	6	27	21	16	16
7	10	1	1	11	1	0	4

Average TAT : 13.71
Average WT : 9.86
~~Average~~ R

```
| P1 | P5 | P7 | P1 | P2 | P3 | P4 | P6 |
0    5    11   12   15   17   21   22   27
```

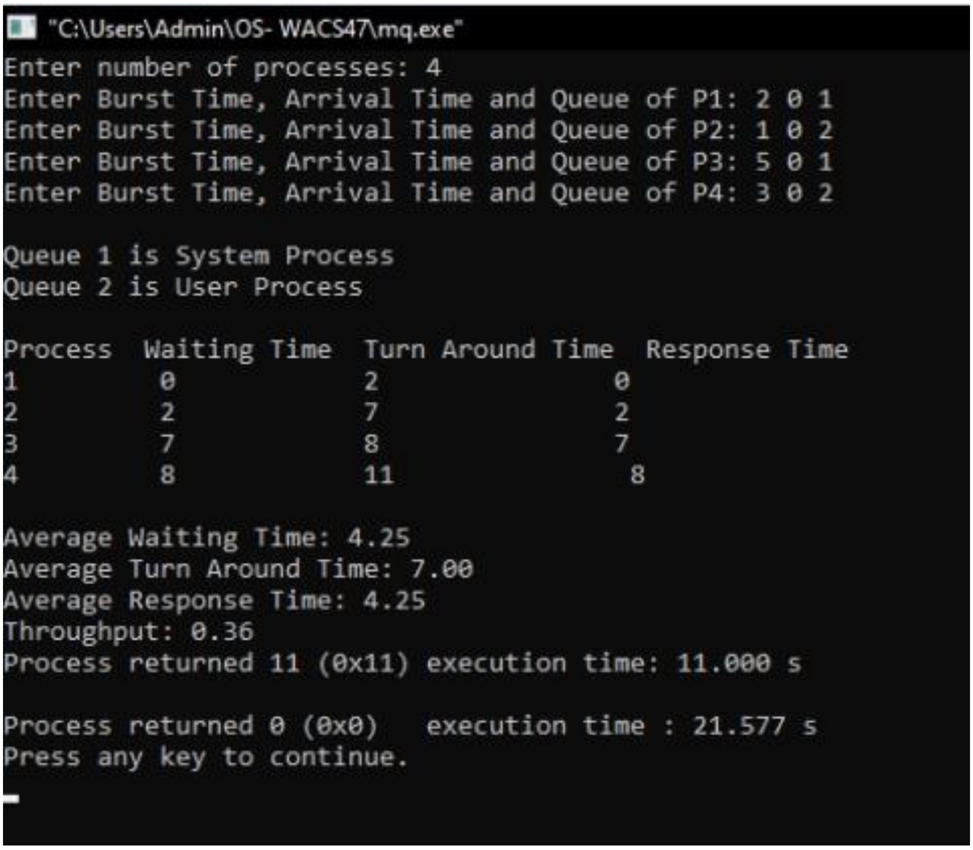
3.Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for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

```
#include <stdio.h>
#define MAX_PROCESSES 10
#define TIME_QUANTUM 2
typedef struct {
    int burst_time, arrival_time, queue_type, waiting_time, turnaround_time,
response_time, remaining_time;
} Process;
void round_robin(Process processes[], int n, int time_quantum, int *time) {
    int done, i;
    do {
        done = 1;
        for (i = 0; i < n; i++) {
            if (processes[i].remaining_time > 0) {
                done = 0;
                if (processes[i].remaining_time > time_quantum) {
                    *time += time_quantum;
                    processes[i].remaining_time -= time_quantum;
                } else {
                    *time += processes[i].remaining_time;
                    processes[i].waiting_time = *time - processes[i].arrival_time -
processes[i].burst_time;
                    processes[i].turnaround_time = *time - processes[i].arrival_time;
                    processes[i].response_time = processes[i].waiting_time;
                    processes[i].remaining_time = 0;
                }
            }
        }
    } while (!done);
}
void fcfs(Process processes[], int n, int *time) {
    for (int i = 0; i < n; i++) {
        if (*time < processes[i].arrival_time) {
            *time = processes[i].arrival_time;
        }
        processes[i].waiting_time = *time - processes[i].arrival_time;
        processes[i].turnaround_time = processes[i].waiting_time +
processes[i].burst_time;
        processes[i].response_time = processes[i].waiting_time;
        *time += processes[i].burst_time;
    }
}
```

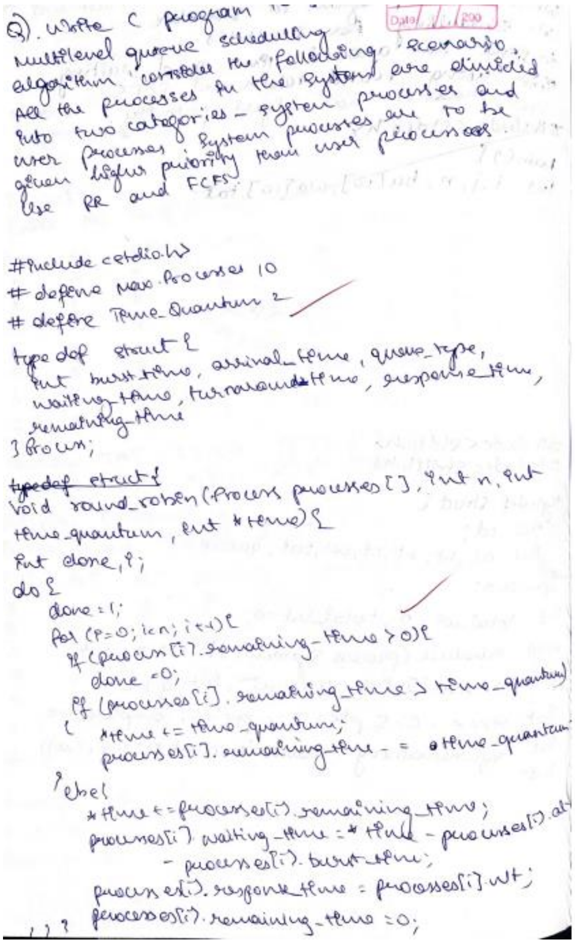
```
int main() {
   Process processes[MAX_PROCESSES], system_queue[MAX_PROCESSES],
user_queue[MAX_PROCESSES];
   int n, sys_count = 0, user_count = 0, time = 0;
   float avg_waiting = 0, avg_turnaround = 0, avg_response = 0, throughput;
   printf("Enter number of processes: ");
   scanf("%d", &n);
   for (int i = 0; i < n; i++) {
      printf("Enter Burst Time, Arrival Time and Queue of P%d: ", i + 1);
      scanf("%d %d %d", &processes[i].burst_time, &processes[i].arrival_time,
&processes[i].queue_type);
      processes[i].remaining_time = processes[i].burst_time;

      if (processes[i].queue_type == 1) {
         system_queue[sys_count++] = processes[i];
      } else {
         user_queue[user_count++] = processes[i];
      }
   }
   for (int i = 0; i < user_count - 1; i++) {
      for (int j = 0; j < user_count - i - 1; j++) {
         if (user_queue[j].arrival_time > user_queue[j + 1].arrival_time) {
            Process temp = user_queue[j];
            user_queue[j] = user_queue[j + 1];
            user_queue[j + 1] = temp;
         }
      }
   }
   printf("\nQueue 1 is System Process\nQueue 2 is User Process\n");
   round_robin(system_queue, sys_count, TIME_QUANTUM, &time);
   fcfs(user_queue, user_count, &time);

   printf("\nProcess  Waiting Time  Turn Around Time  Response Time\n");

   for (int i = 0; i < sys_count; i++) {
      avg_waiting += system_queue[i].waiting_time;
      avg_turnaround += system_queue[i].turnaround_time;
      avg_response += system_queue[i].response_time;
      printf("%d        %d           %d              %d\n", i + 1,
system_queue[i].waiting_time, system_queue[i].turnaround_time,
system_queue[i].response_time);
   }

   for (int i = 0; i < user_count; i++) {
      avg_waiting += user_queue[i].waiting_time;
```

```
            avg_turnaround += user_queue[i].turnaround_time;
            avg_response += user_queue[i].response_time;
            printf("%d        %d          %d              %d\n", i + 1 + sys_count,
    user_queue[i].waiting_time, user_queue[i].turnaround_time,
    user_queue[i].response_time);
        }

        avg_waiting /= n;
        avg_turnaround /= n;
        avg_response /= n;
        throughput = (float)n / time;

        printf("\nAverage Waiting Time: %.2f", avg_waiting);
        printf("\nAverage Turn Around Time: %.2f", avg_turnaround);
        printf("\nAverage Response Time: %.2f", avg_response);
        printf("\nThroughput: %.2f", throughput);
        printf("\nProcess returned %d (0x%d) execution time: %.3f s\n", time, time,
    (float)time);

        return 0;
}
```

Output:



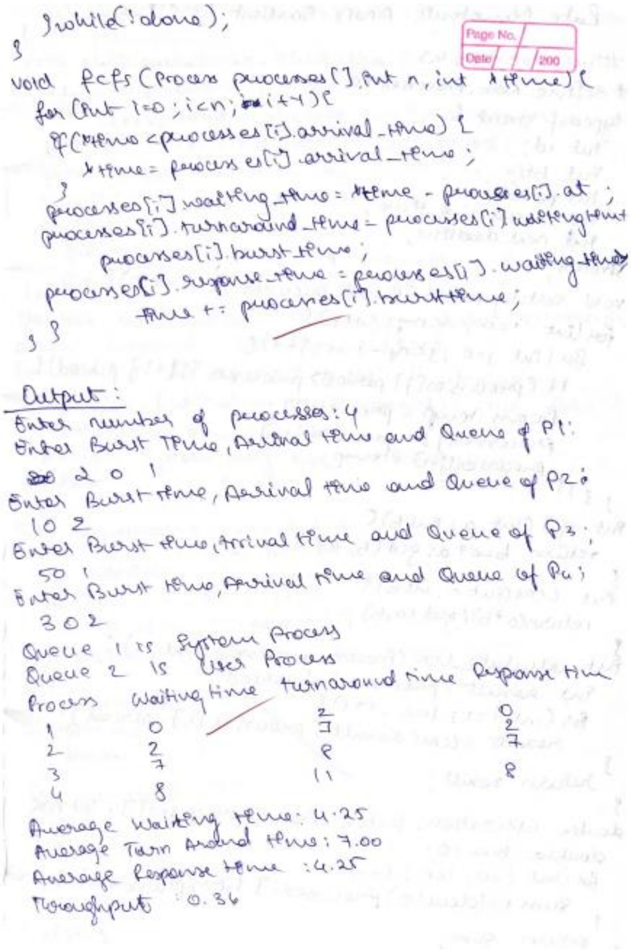
```
Enter number of processes: 4
Enter Burst Time, Arrival Time and Queue of P1: 2 0 1
Enter Burst Time, Arrival Time and Queue of P2: 1 0 2
Enter Burst Time, Arrival Time and Queue of P3: 5 0 1
Enter Burst Time, Arrival Time and Queue of P4: 3 0 2

Queue 1 is System Process
Queue 2 is User Process

Process  Waiting Time  Turn Around Time  Response Time
1        0             2                 0
2        2             7                 2
3        7             8                 7
4        8             11                 8

Average Waiting Time: 4.25
Average Turn Around Time: 7.00
Average Response Time: 4.25
Throughput: 0.36
Process returned 11 (0x11) execution time: 11.000 s

Process returned 0 (0x0)   execution time : 21.577 s
Press any key to continue.
```



Date / / 200

Q). Write C program multilevel queue scheduling algorithm consider 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 RR and FCFS

```
#include <stdio.h>
# define Max Processes 10
# define Time_Quantum 2

typedef struct {
  int burst time, arrival_time, queue_type,
  waiting_time, turnaround time, response_time,
  remaining_time
} Process;

~~typedef struct {~~
void round_robin(Process processes[], int n, int
time_quantum, int *time) {
int done, i;
do {
    done = 1;
    for (i=0; i<n; i++) {
      if (process[i].remaining_time > 0) {
        done = 0;
      if (processes[i]. remaining_time > time_quantum)
      {   *time += time_quantum;
          processes[i]. remaining_time -= time_quantum
      } else {
        *time += processes[i]. remaining_time;
        processes[i]. waiting_time = *time - processes[i].at
                  - processes[i]. burst_time;
        processes[i]. response_time = processes[i].wt;
  } } }  processes[i]. remaining_time = 0;
```



Page No. / Date / / 200

```
  } while(!done);
}
void fcfs(Process processes[], int n, int *time) {
  for (int i=0; i<n; i++) {
   if(*time < processes[i].arrival_time) {
      *time = processes[i].arrival_time;
   }
  processes[i].waiting_time = *time - processes[i].at ;
 processes[i].turnaround_time = processes[i].waiting time +
         processes[i].burst time;
 processes[i].response_time = processes[i].waiting_time;
          time += processes[i].burst time;
 } }
```

Output:

```
Enter number of processes: 4
Enter Burst Time, Arrival time and Queue of P1:
 ~~20~~ 2 0 1
Enter Burst time, Arrival time and Queue of P2:
 10 2
Enter Burst time, Arrival time and Queue of P3:
 5 0 1
Enter Burst time, Arrival time and Queue of P4:
 3 0 2
Queue 1 is System Process
Queue 2 is User Process
Process   Waiting time    turnaround time   Response time
                              2                 0
  1          0                7                 2
  2          2                8                 7
  3          7               11                 8
  4          8
Average Waiting time: 4.25
Average Turn Around time: 7.00
Average Response time  : 4.25
Throughput  : 0.36
```

4.Write a C program to simulate Real-Time CPU Scheduling

algorithms: a) Rate- Monotonic b) Earliest-deadline First

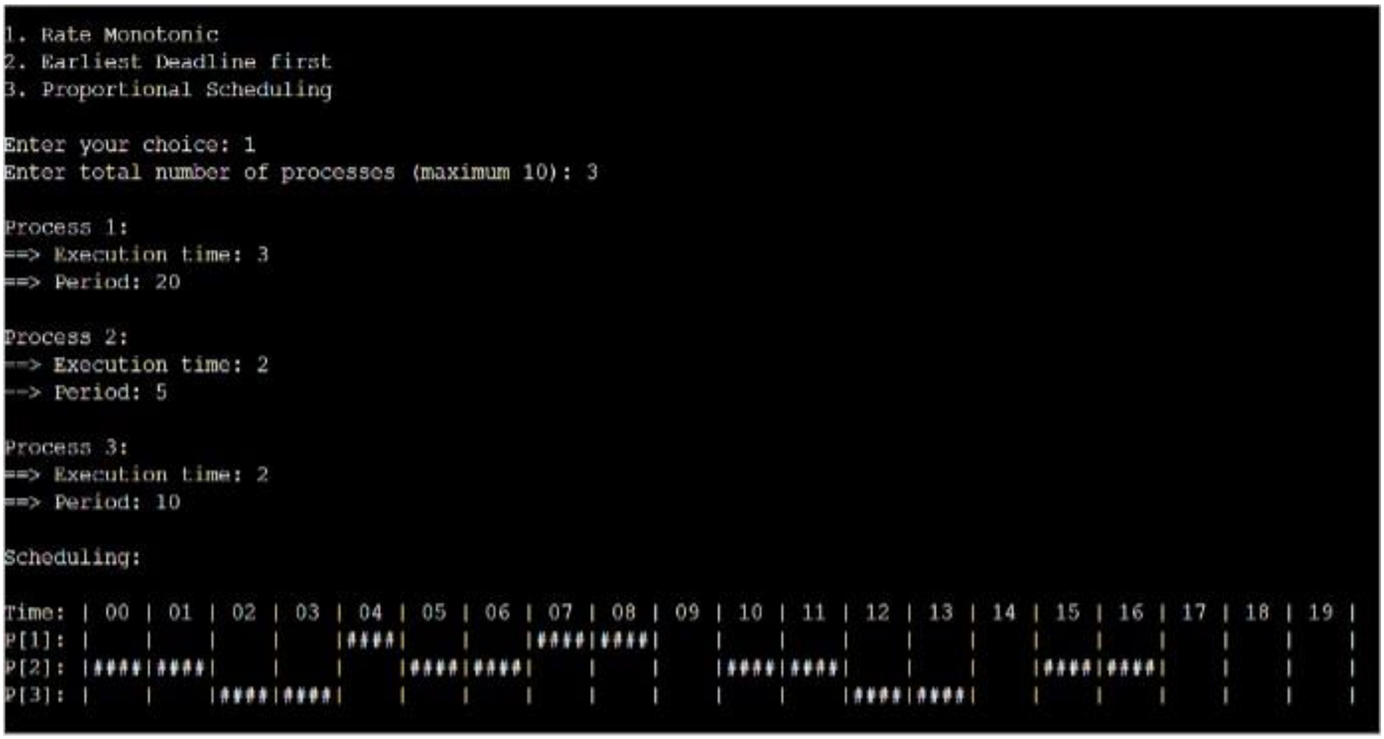
```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <stdbool.h>

#define MAX_PROCESS 10

int num_of_process;

int execution_time[MAX_PROCESS], period[MAX_PROCESS],

    remain_time[MAX_PROCESS], deadline[MAX_PROCESS],

    remain_deadline[MAX_PROCESS];
```

```
void get_process_info() {
    printf("Enter total number of processes (maximum %d): ", MAX_PROCESS);
    scanf("%d", &num_of_process);
    if (num_of_process < 1) {
        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]);
    }
}
int max(int a, int b, int c) {
    int max;
    if (a >= b && a >= c)
        max = a;
    else if (b >= a && b >= c)
        max = b;
    else if (c >= a && c >= b)
```

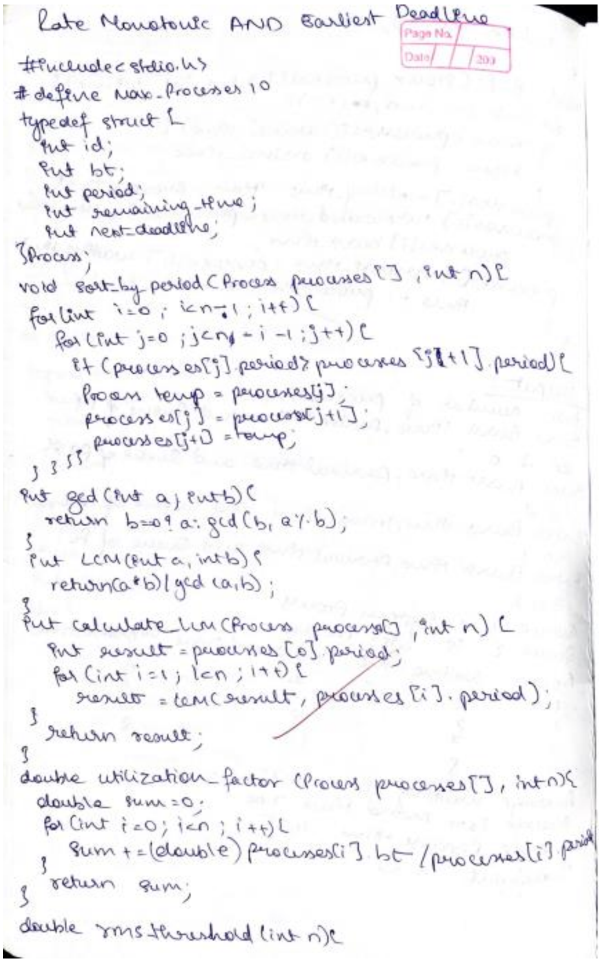
```
        max = c;
    return max;
}
void print_schedule(int process_list[], int cycles) {
    printf("\nScheduling:\n\n");
    printf("Time: ");
    for (int i = 0; i < cycles; i++) {
        if (i < 10)
            printf("| 0%d ", i);
        else
            printf("| %d ", i);
    }
    printf("|\n");
 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) {
    int process_list[100] = {0}, min = 999, next_process = 0;
    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;
    float m = n * (pow(2, 1.0 / n) - 1);
    if (utilization > m) {
        printf("\nGiven problem is not schedulable under the said scheduling
algorithm.\n");
    }
    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;
            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 observation_time;
    get_process_info();
    observation_time = max(period[0], period[1], period[2]);
    rate_monotonic(observation_time);
    return 0;
}
```

Output:



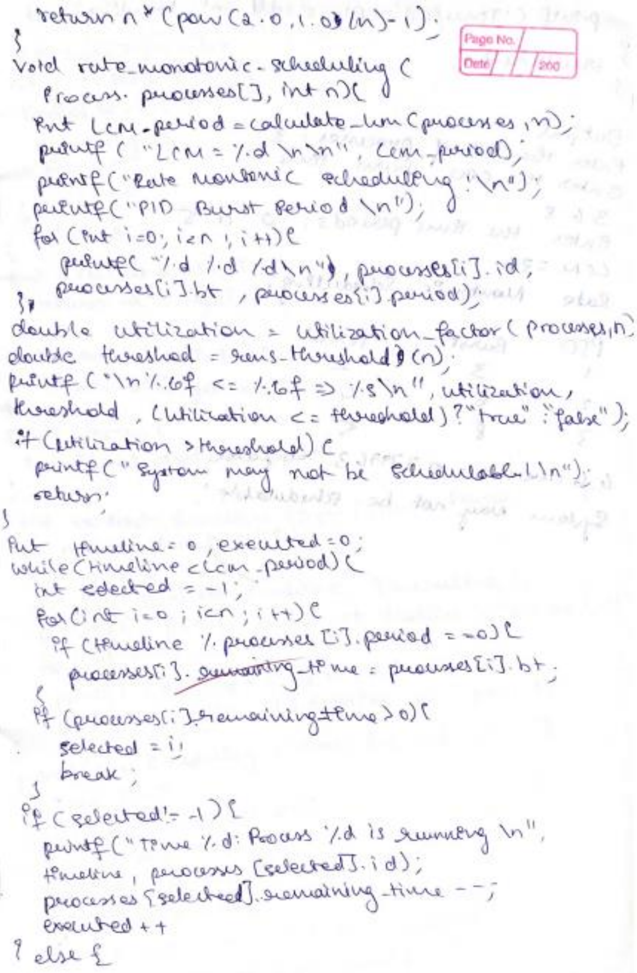
```
1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

Enter your choice: 1
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 3
==> Period: 20

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

Process 3:
==> Execution time: 2
==> Period: 10

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
P[1]: |    |    |    |    |####|    |    |####|####|    |    |    |    |    |    |    |    |    |    |    |
P[2]: |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |
P[3]: |    |    |####|####|    |    |    |    |    |    |    |    |####|####|    |    |    |    |    |    |
```



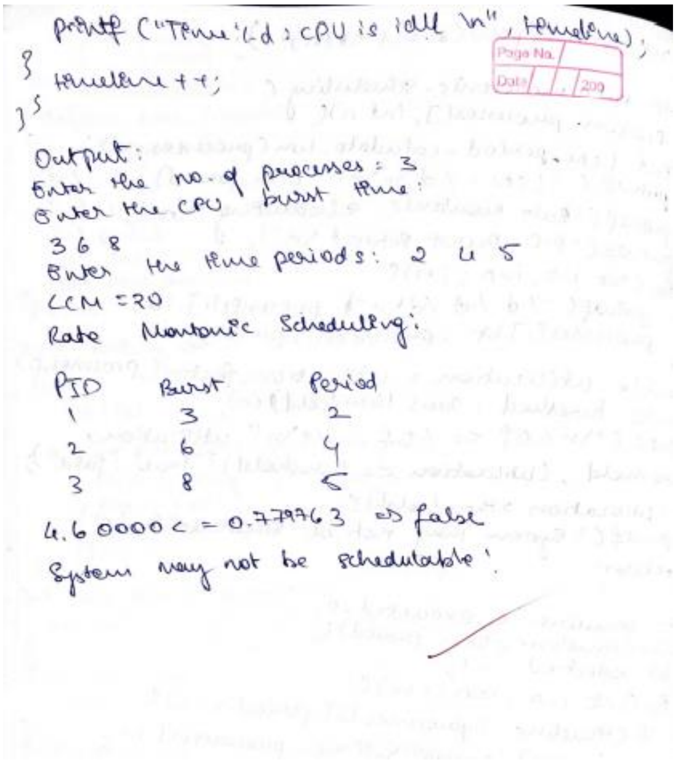
Rate Monotonic AND Earliest Deadline

Page No.
Date / / 200

```
#include<stdio.h>
#define Max_Processes 10
typedef struct {
  int id;
  int bt;
  int period;
  int remaining_time;
  int next_deadline;
}Process;
void sort_by_period(Process processes[], int n){
 for(int i=0; i<n-1; i++){
   for(int j=0; j<n-i-1; j++){
     if(processes[j].period> processes[j+1].period){
      Process temp = processes[j];
      processes[j] = process[j+1];
       processes[j+1] = temp;
 } } }}
int gcd(int a, int b){
  return b==0? a: gcd(b, a%b);
}
int LCM(int a, int b){
  return(a*b)/gcd(a,b);
}
int calculate_lcm(Process process[], int n) {
  int result = processes[0].period;
  for(int i=1; i<n; i++){
     result = lcm(result, processes[i].period);
  }
  return result;
}
double utilization_factor(Process processes[], int n){
 double sum=0;
 for(int i=0; i<n; i++){
   sum += (double) processes[i].bt / processes[i].period
 }
 return sum;
}
double rms_threshold(int n){
```



```
 return n*(pow(2.0,1.0/n)-1);
}
void rate_monotonic_scheduling(
  Process processes[], int n){
 int lcm_period = calculate_lcm(processes, n);
 printf("LCM = %d \n\n", lcm_period);
 printf("Rate Monotonic Scheduling:\n");
 printf("PID Burst Period \n");
 for(int i=0; i<n; i++){
   printf("%d %d %d\n", processes[i].id,
   processes[i].bt, processes[i].period);
 }
double utilization = utilization_factor(processes,n);
double threshold = rms_threshold(n);
printf("\n%.6f <= %.6f => %s\n", utilization,
threshold, (utilization <= threshold)?"true":"false");
if(utilization >threshold){
  printf("System may not be schedulable.\n");
  return;
}
int timeline=0, executed=0;
while(timeline<lcm_period){
  int selected = -1;
  for(int i=0; i<n; i++){
   if(timeline % processes[i].period ==0){
     processes[i].remaining_time = processes[i].bt;
  }
  if(processes[i].remaining_time>0){
    selected = i;
    break;
  }
  if(selected!=-1){
   printf("Time %d: Process %d is running \n",
   timeline, processes[selected].id);
   processes[selected].remaining_time --;
   executed++
 } else {
```



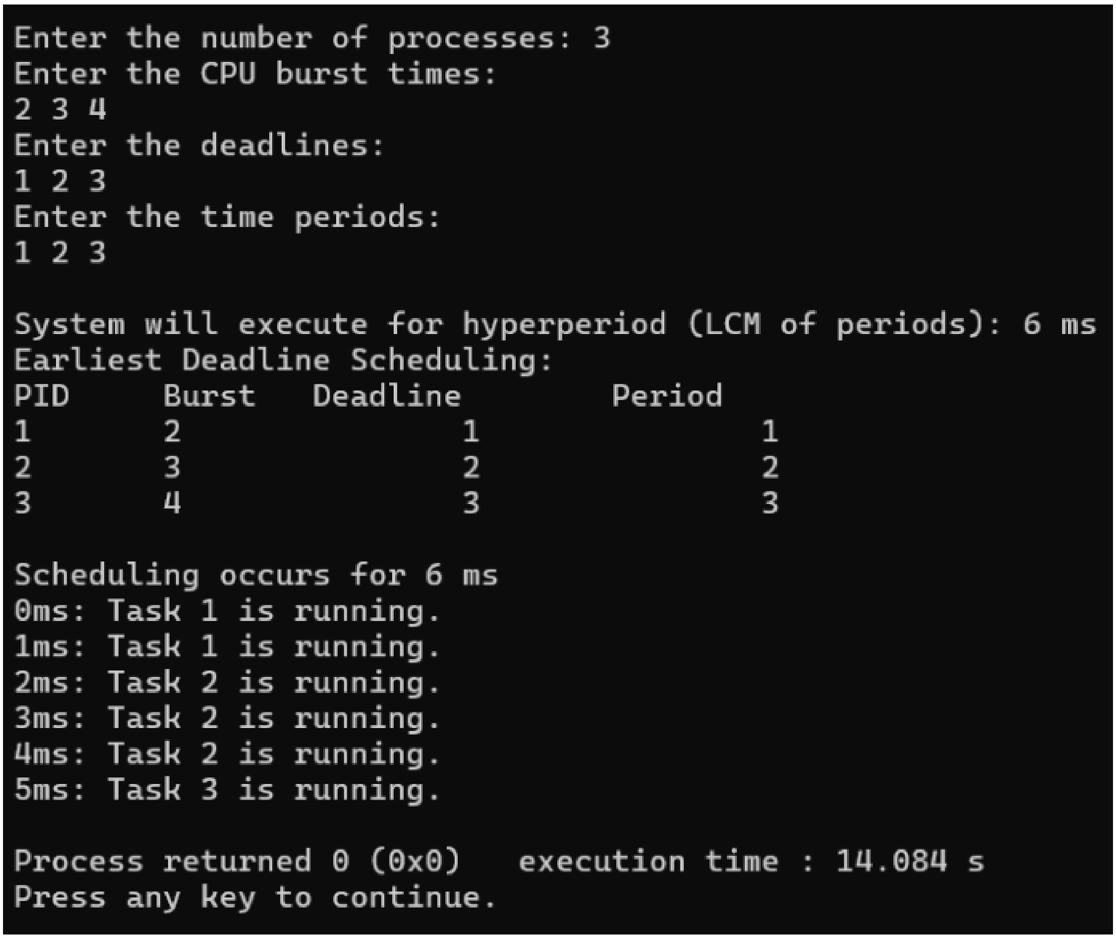
```
printf ("Time %d : CPU is idle \n", timeline);
}
timeline ++;
}}
```

```
Output:
Enter the no. of processes : 3
Enter the CPU burst time:
 3 6 8
Enter the time periods: 2 4 5
LCM =20
Rate Monotonic Scheduling:

 PID    Burst    Period
  1      3        2
  2      6        4
  3      8        5
4.60000 <= 0.779763 => false
System may not be schedulable!
```

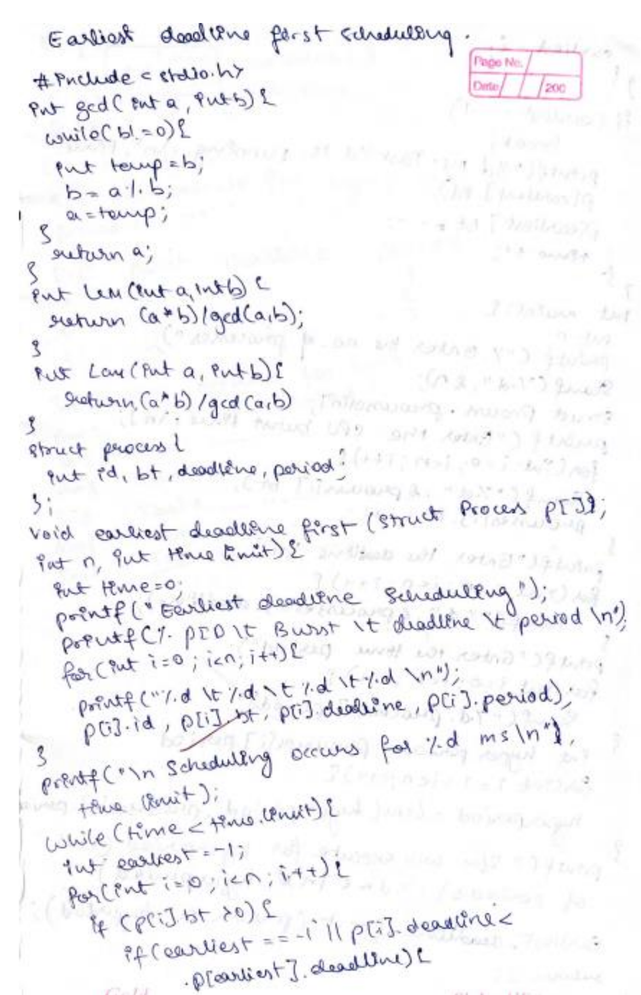
Earliest Deadline

```
#include <stdio.h>

int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int lcm(int a, int b) {
    return (a * b) / gcd(a, b);
}

struct Process {
    int id, burst_time, deadline, period;
};

void earliest_deadline_first(struct Process p[], int n, int time_limit) {
    int time = 0;
    printf("Earliest Deadline Scheduling:\n");
    printf("PID\tBurst\tDeadline\tPeriod\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].burst_time, p[i].deadline, p[i].period);
    }

    printf("\nScheduling occurs for %d ms\n", time_limit);
    while (time < time_limit) {
        int earliest = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].burst_time > 0) {
                if (earliest == -1 || p[i].deadline < p[earliest].deadline) {
                    earliest = i;
                }
            }
        }

        if (earliest == -1) break;

        printf("%dms: Task %d is running.\n", time, p[earliest].id);
        p[earliest].burst_time--;
        time++;
    }
}

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

    struct Process processes[n];
    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].burst_time);
        processes[i].id = i + 1;
    }

    printf("Enter the deadlines:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].deadline);
    }

    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].period);
    }

    int hyperperiod = processes[0].period;
    for (int i = 1; i < n; i++) {
            hyperperiod = lcm(hyperperiod, processes[i].period);
        }

        printf("\nSystem will execute for hyperperiod (LCM of periods): %d ms\n",
    hyperperiod);

        earliest_deadline_first(processes, n, hyperperiod);

        return 0;
}
```

Output:

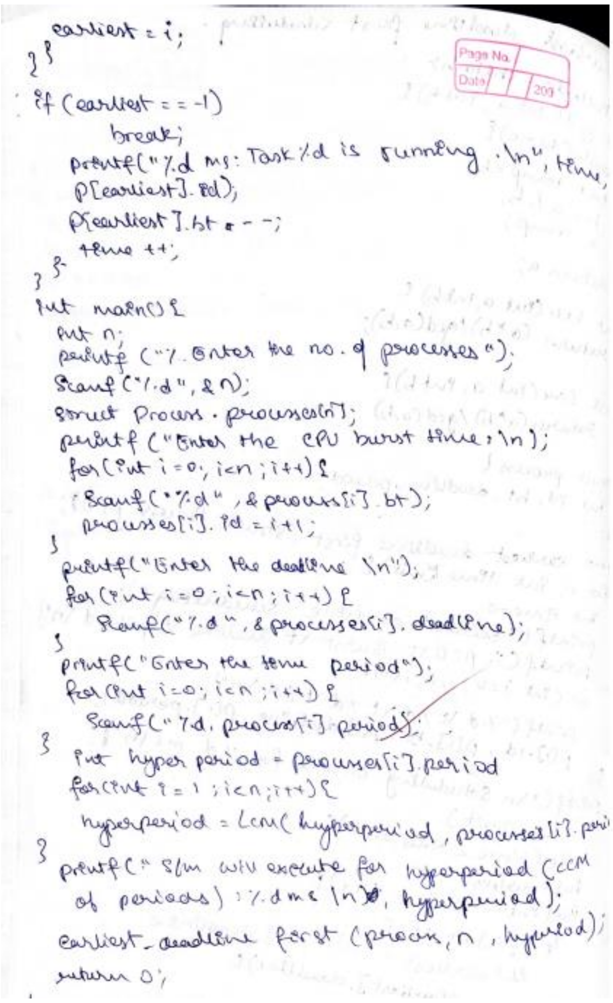


```
Enter the number of processes: 3
Enter the CPU burst times:
2 3 4
Enter the deadlines:
1 2 3
Enter the time periods:
1 2 3

System will execute for hyperperiod (LCM of periods): 6 ms
Earliest Deadline Scheduling:
PID     Burst   Deadline        Period
1       2               1               1
2       3               2               2
3       4               3               3

Scheduling occurs for 6 ms
0ms: Task 1 is running.
1ms: Task 1 is running.
2ms: Task 2 is running.
3ms: Task 2 is running.
4ms: Task 2 is running.
5ms: Task 3 is running.

Process returned 0 (0x0)   execution time : 14.084 s
Press any key to continue.
```

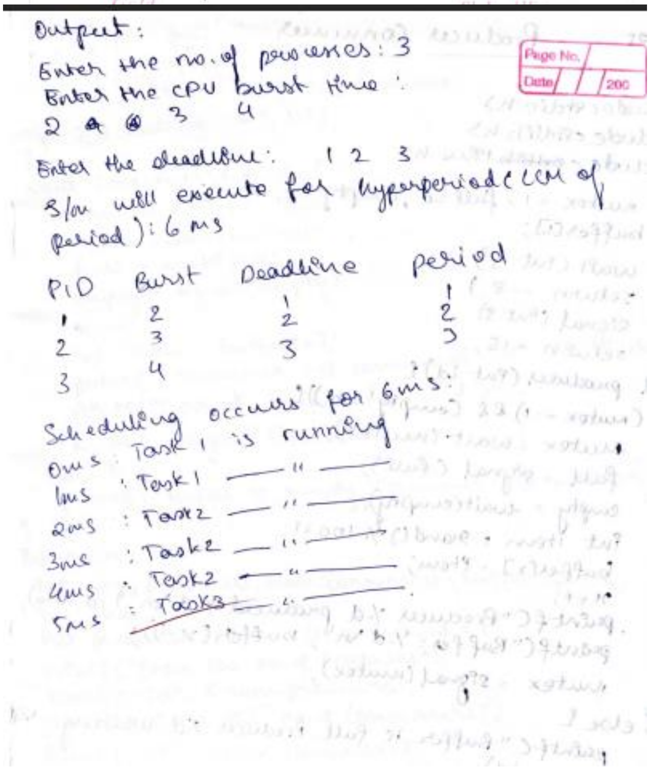


Earliest deadline first Scheduling.

```
#include <stdio.h>
int gcd(int a, int b){
 while(b!=0){
  int temp=b;
   b=a%b;
   a=temp;
 }
  return a;
}
int Lcm(int a,int b) {
  return (a*b)/gcd(a,b);
}
int Lcm(int a, int b){
   return (a*b)/gcd(a,b)
}
struct process {
 int id, bt, deadline, period;
};
void earliest deadline first (struct Proces p[]);
int n, int time limit){
 int time=0;
 printf("Earliest deadline Scheduling");
 printf("% PID \t Burst \t deadline \t period \n");
 for(int i=0; i<n; i++){
   printf("%d \t %d \t %d \t %d \n");
   p[i].id, p[i].bt, p[i].deadline, p[i].period);
 }
 printf("\n Scheduling occurs for %d ms \n"),
   time limit);
 while(time < time.limit){
  int earliest=-1;
  for(int i=0; i<n; i++){
    if (p[i].bt >0) {
      if(earliest == -1 || p[i].deadline<
        .p[earliest].deadline){
```



```
  earliest = i;
 }}
 if (earliest == -1)
      break;
   printf("%d ms: Task %d is running.\n", time,
   p[earliest].id);
   p[earliest].bt --;
   time ++;
 }
}
int main(){
 int n;
 printf("% Enter the no. of processes");
 Scanf("%d", &n);
 struct Process. processes[n];
 printf("Enter the CPU burst time:\n);
  for(int i=0; i<n; i++){
   Scanf("%d", &process[i].bt);
   processes[i]. id = i+1;
 }
 printf("Enter the deadline \n");
 for(int i=0; i<n; i++) {
   Scanf("%d", &processes[i]. deadline);
 }
 printf("Enter the time period");
 for(int i=0; i<n; i++) {
   Scanf(" %d, process[i].period);
 }
 int hyper period = processes[i].period
 for(int i=1; i<n; i++){
  hyperperiod = Lcm( hyperperiod, processes[i].peri
 }
 printf(" S/m will execute for hyperperiod (LCM
  of periods) : %d ms \n", hyperperiod);
earliest_deadline first (process, n, hyperiod);
return 0;
```



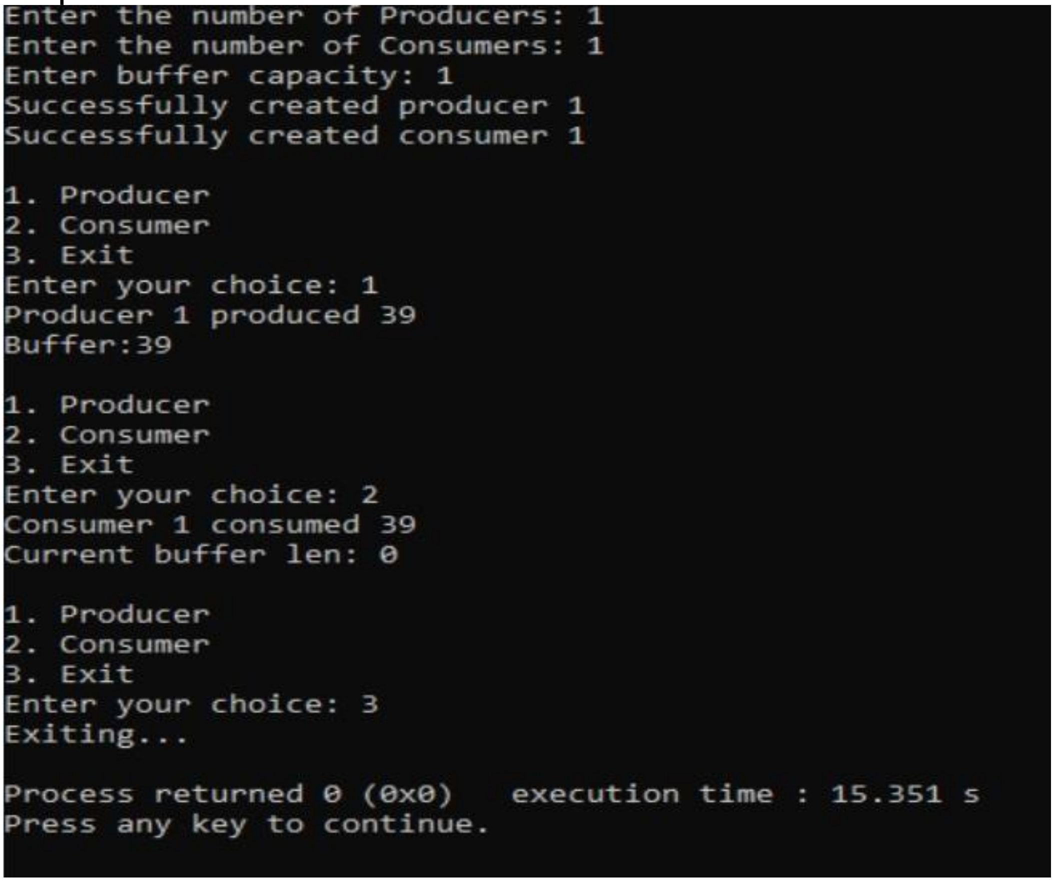
Output:

```
Enter the no. of processes: 3
Enter the CPU burst time :
2  3  4
Enter the deadline:   1  2  3
S/m will execute for hyperperiod (LCM of
period ): 6 ms
PID   Burst   Deadline   period
                            1
 1     2        1           2
 2     3        2           3
 3     4        3
Scheduling occurs for 6ms:
0ms : Task 1 is running
1ms : Task 1 —— " ——
2ms : Task 2 —— " ——
3ms : Task 2 —— " ——
4ms : Task 2 —— " ——
5ms : Task 3 —— " ——
```

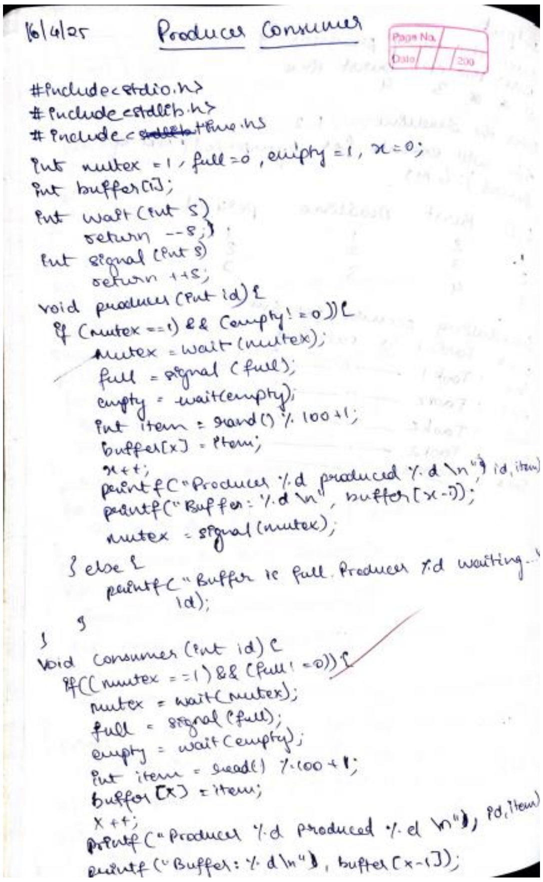
5. Write a C program to simulate producer-consumer problem using semaphores

```
#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);

int main() {
    int n;
    printf("\n1. Producer\n2. Consumer\n3. Exit");
    while(1) {
        printf("\nEnter your choice: ");
        scanf("%d", &n);
        switch(n) {
            case 1:
                if((mutex == 1) && (empty != 0))
                    producer();


                else
                    printf("Buffer is full!!\n");
                break;
            case 2:
                if((mutex == 1) && (full != 0))
                    consumer();
                else
                    printf("Buffer is empty!!\n");
                break;
            case 3:
                exit(0);
                break;
        }
    }
    return 0;
}

int wait(int s) {
    return (--s);
}

int signal(int s) {
    return (++s);
}

void producer() {
    mutex = wait(mutex);
    full = signal(full);
    empty = wait(empty);
    x++;
    printf("\nProducer produces the item %d\n", x);
    mutex = signal(mutex);
}

void consumer() {
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf("\nConsumer consumes item %d\n", x);
    x--;
    mutex = signal(mutex);
}
```

Output:



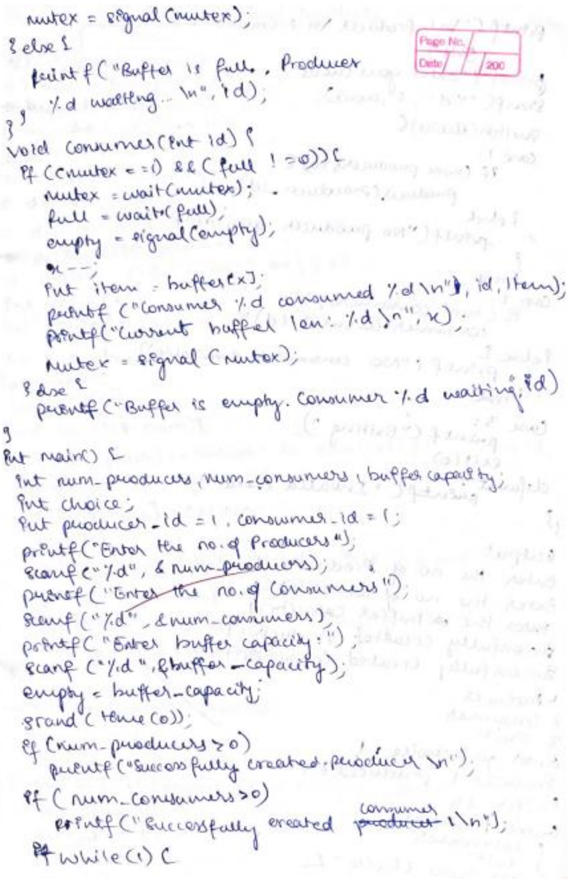
```
Enter the number of Producers: 1
Enter the number of Consumers: 1
Enter buffer capacity: 1
Successfully created producer 1
Successfully created consumer 1

1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer 1 produced 39
Buffer:39

1. Producer
2. Consumer
3. Exit
Enter your choice: 2
Consumer 1 consumed 39
Current buffer len: 0

1. Producer
2. Consumer
3. Exit
Enter your choice: 3
Exiting...

Process returned 0 (0x0)   execution time : 15.351 s
Press any key to continue.
```



16/4/25 Producer Consumer

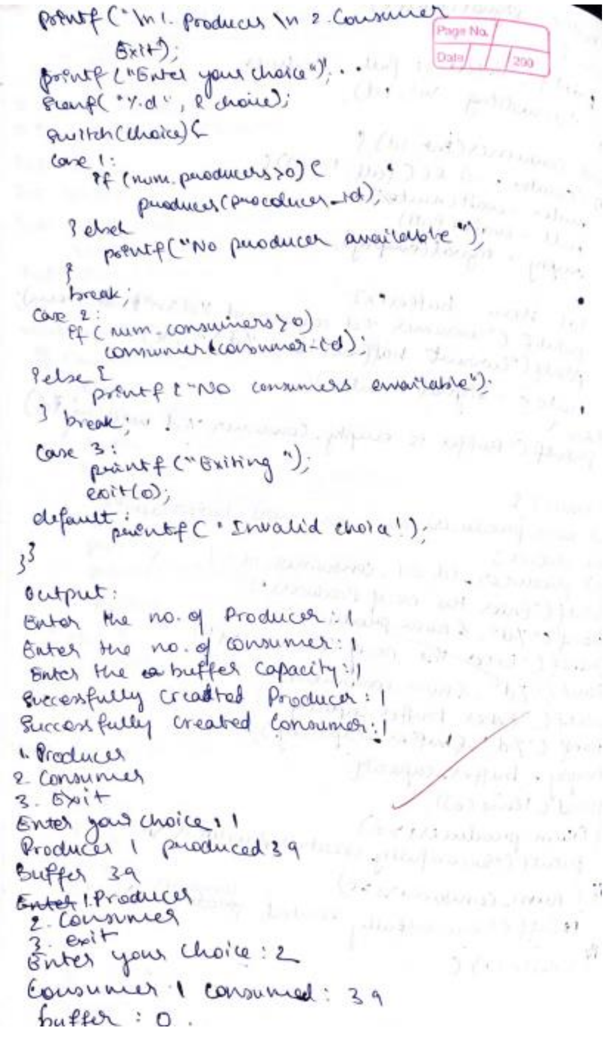
Page No. Date / / 200

```
#include<stdio.h>
#include<stdlib.h>
#include<~~stdlib~~time.h>
int mutex =1, full=0 ,empty =1, x=0;
int buffer[1];
int wait (int s)
     return --s;}
int signal (int s)
     return ++s;
void producer (int id) {
  if (mutex ==1) && (empty! =0)){
     mutex =wait (mutex);
     full = signal (full);
     empty = wait(empty);
     int item = rand () % 100+1;
     buffer[x] = item;
     x++;
     printf("Producer %d produced %d \n", id, item)
     printf(" Buffer: %d \n", buffer[x-1]);
     mutex = signal (mutex);

  } else {
     printf(" Buffer is full. Producer %d waiting..."
              id);
    }
}
void consumer (int id) {
  if((mutex ==1)&& (full! =0)){
    mutex = wait(mutex);
    full = signal (full);
    empty = wait (empty);
    int item = read() %100 + 1;
    buffer [x] = item;
    x ++;
   printf ("Producer %d produced %d \n"), id, item)
   printf ("Buffer: % d\n"), buffer [x-1]);
```



Page No. Date / / 200

```
 mutex = signal (mutex);
} else {
  printf ("Buffer is full. Producer
   %d waiting.. \n", id);
}}
void consumer(int id) {
 if (mutex ==1) && (full !=0)){
   mutex =wait(mutex);
   full = wait(full);
   empty = signal(empty);
   x--;
   int item = buffer[x];
   printf ("Consumer %d consumed %d \n"), id, item);
   printf("Current buffer len: %d \n", x);
   mutex = signal (mutex);
 } else {
  printf ("Buffer is empty. Consumer %d waiting, id)
}
int main() {
 int num_producers, num_consumers, buffer capacity;
 int choice;
 int producer_id =1, consumer_id = 1;
 printf("Enter the no. of Producers");
 scanf("%d", & num_producers);
 printf("Enter the no. of Consumers");
 scanf("%d", &num_consumers)
 printf("Enter buffer capacity:")
 scanf ("%d", &buffer_capacity);
 empty = buffer_capacity;
 srand ( time (0));
 if (num_producers>0)
   printf ("Successfully created producer \n");
 if (num_consumers>0)
    printf("Successfully created ~~producer~~ consumer 1\n");
  while(1) {
```

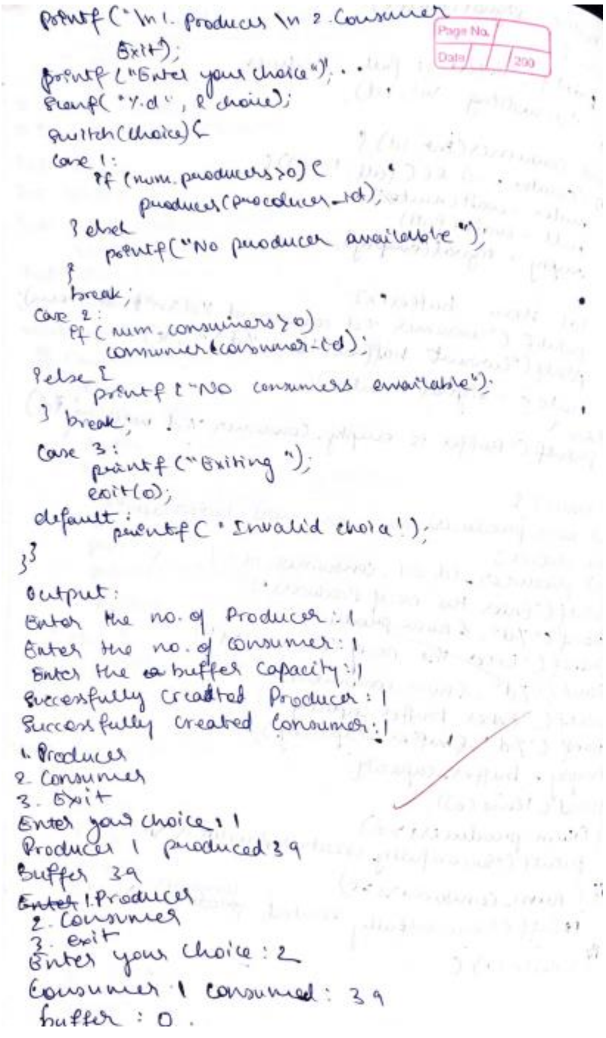


Page No. Date / / 200

```
printf("\n1. Producer \n 2 Consumer
        Exit");
printf ("Enter your choice");
scanf( "%d", & choice);
switch(choice){
 case 1:
     if (num.producers>0) {
          producer(producer_id);
   } else
      printf("No producer available");
   }
   break;
 case 2:
    if (num_consumers>0)
      consumer(consumer_id);
 } else {
     printf ("No consumers available");
  } break;
 case 3:
     printf ("Exiting ");
     exit(0);
 default:
     printf (' Invalid choice!);
}}
```

Output:

```
Enter the no. of Producer: 1
Enter the no. of Consumer: 1
Enter the ~~a~~ buffer capacity: 1
Successfully created Producer: 1
Successfully created Consumer: 1
1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer 1 produced 39
Buffer 39
~~Enter~~ 1.Producer
 2. Consumer
 3. exit
Enter your choice: 2
Consumer 1 consumed: 39
 buffer : 0 .
```



Page No. Date / / 200

```
printf("\n1. Producer \n 2 Consumer
        Exit");
printf ("Enter your choice");
scanf( "%d", & choice);
switch(choice){
 case 1:
     if (num.producers>0) {
          producer(producer_id);
   } else
      printf("No producer available");
   }
   break;
 case 2:
    if (num_consumers>0)
      consumer(consumer_id);
 } else {
     printf ("No consumers available");
  } break;
 case 3:
     printf ("Exiting ");
     exit(0);
 default:
     printf (' Invalid choice!);
}}
```

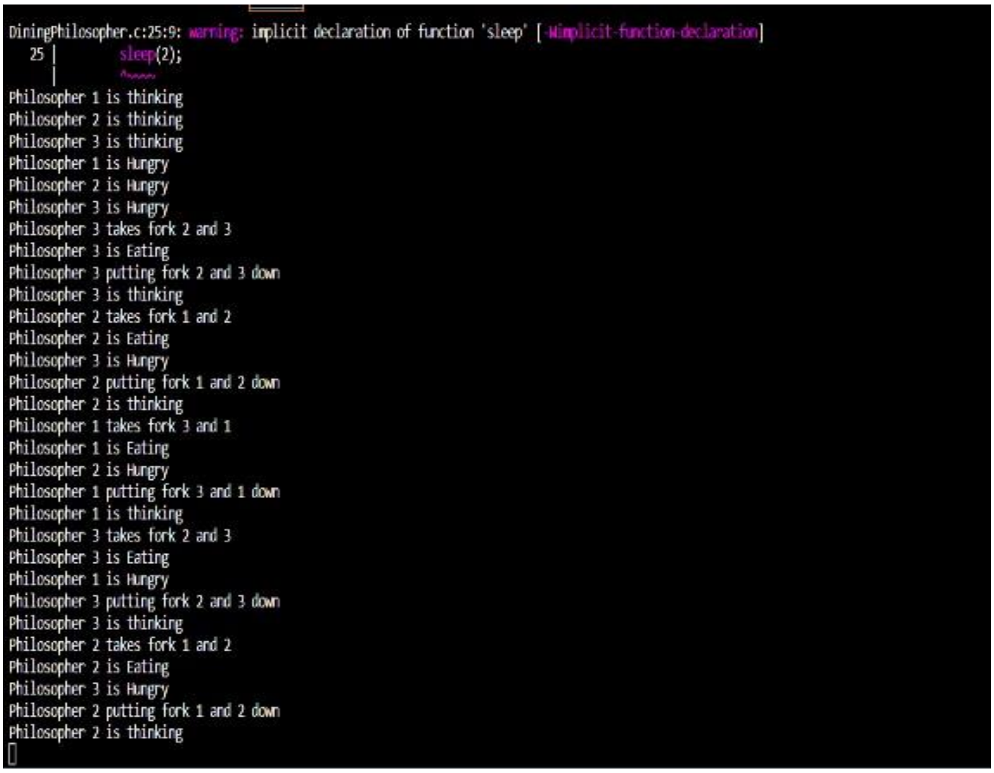
Output:

```
Enter the no. of Producer: 1
Enter the no. of Consumer: 1
Enter the ~~a~~ buffer capacity: 1
Successfully created Producer: 1
Successfully created Consumer: 1
1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer 1 produced 39
Buffer 39
~~Enter~~ 1.Producer
 2. Consumer
 3. exit
Enter your choice: 2
Consumer 1 consumed: 39
 buffer : 0 .
```

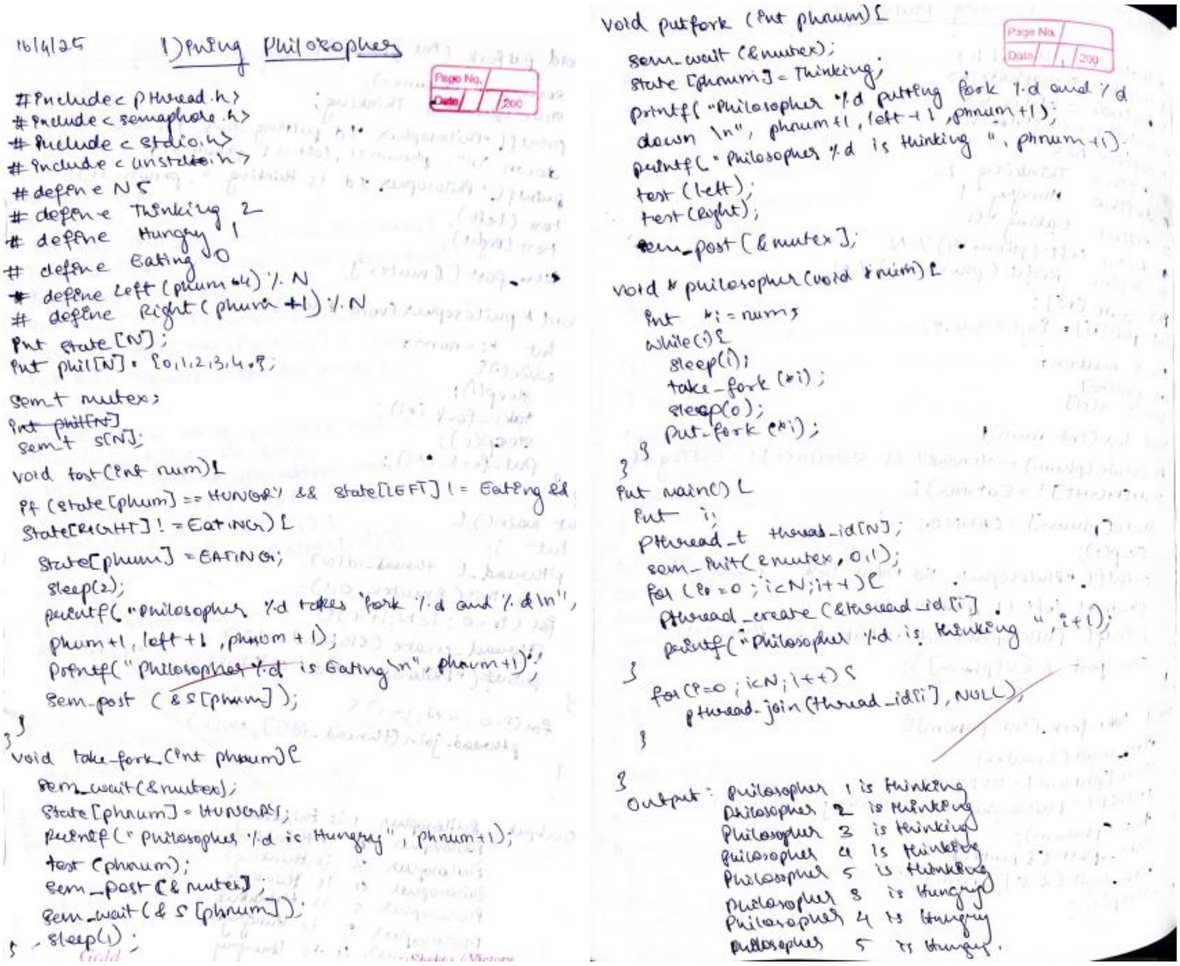
6. Write a C program to simulate the concept of Dining Philosophers problem.

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>  // For usleep

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (i + 4) % N
#define RIGHT (i + 1) % N

int state[N];
int phil[N] = {0, 1, 2, 3, 4};
sem_t mutex;
sem_t S[N];

void test(int i) {
    if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
        state[i] = EATING;
        usleep(2000000);  // Simulate eating time (2 seconds)
        printf("Philosopher %d takes fork %d and %d\n", i + 1, LEFT + 1, i + 1);
        printf("Philosopher %d is Eating\n", i + 1);
        sem_post(&S[i]);
    }
}

void take_fork(int i) {
    sem_wait(&mutex);
    state[i] = HUNGRY;
    printf("Philosopher %d is Hungry\n", i + 1);
    test(i);
    sem_post(&mutex);
    sem_wait(&S[i]);
    usleep(1000000);  // Simulate thinking time (1 second)
}


void put_fork(int i) {
    sem_wait(&mutex);
    state[i] = THINKING;
    printf("Philosopher %d putting fork %d and %d down\n", i + 1, LEFT + 1, i + 1);
    printf("Philosopher %d is thinking\n", i + 1);
    test(LEFT);
    test(RIGHT);
    sem_post(&mutex);
}
void* philosopher(void* num) {
    while (1) {
        int* i = num;
        usleep(1000000);  // Simulate thinking before trying to eat
        take_fork(*i);
        usleep(1000000);  // Simulate time spent eating
        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);
    }

    return 0;
}
```

Output:



```
DiningPhilosopher.c:25:9: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
   25 |         sleep(2);
      |         ^~~~~
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 3 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 1 putting fork 3 and 1 down
Philosopher 1 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 1 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 3 is Hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
```



16/4/25 1) Dining Philosophers

Page No. Date 200

```
#include<pthread.h>
#include<semaphore.h>
#include<stdio.h>
#include<unistd.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;
~~int phil[N]~~
sem_t s[N];
void test(int num){
 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", phnum+1);
   test (phnum);
   sem_post(&mutex);
   sem_wait(& s [phnum]);
   sleep(1);
}
```

Page No. Date 200

```
void putfork (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 ", phnum+1).
  test (left);
  test (Right);
  sem_post (&mutex);

void * philosopher(void *num){
  int *i = num;
  while(1){
    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++){
    pthread_create (&thread_id[i]
    printf(" Philosopher %d is thinking ", 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 4 is Hungry
Philosopher 5 is Hungry.

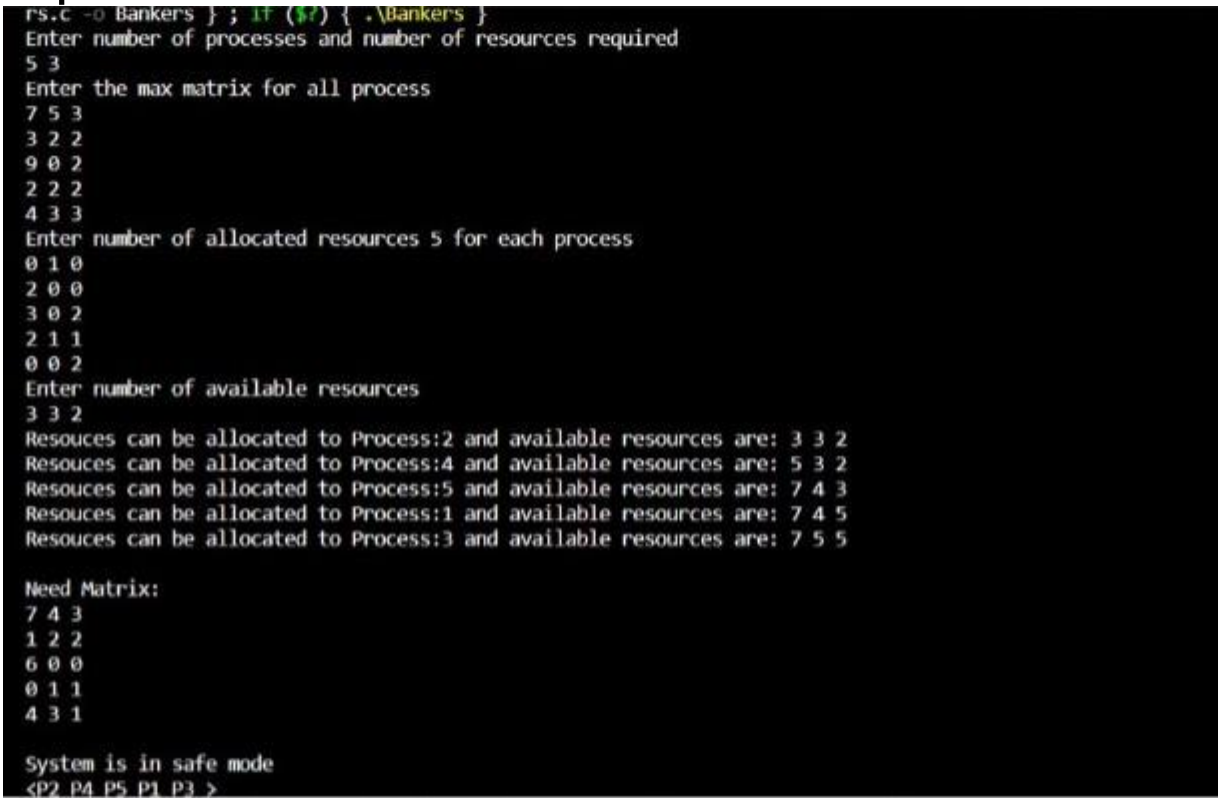
7. Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

```
#include <stdio.h>
int main() {
    int n, m, i, j, k;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resources: ");
    scanf("%d", &m);
    int allocation[n][m];
    int max[n][m];
    int available[m];
    int need[n][m];
    int finish[n], safeSeq[n], index = 0;
    printf("Enter the Allocation Matrix:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            scanf("%d", &allocation[i][j]);
        }
    }
    printf("Enter the MAX Matrix:\n");
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            scanf("%d", &max[i][j]);
        }
    }
    printf("Enter the Available Resources:\n");
    for (i = 0; i < m; i++)
    {
        scanf("%d", &available[i]);
    }
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
        {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
    for (i = 0; i < n; i++)
    {
```

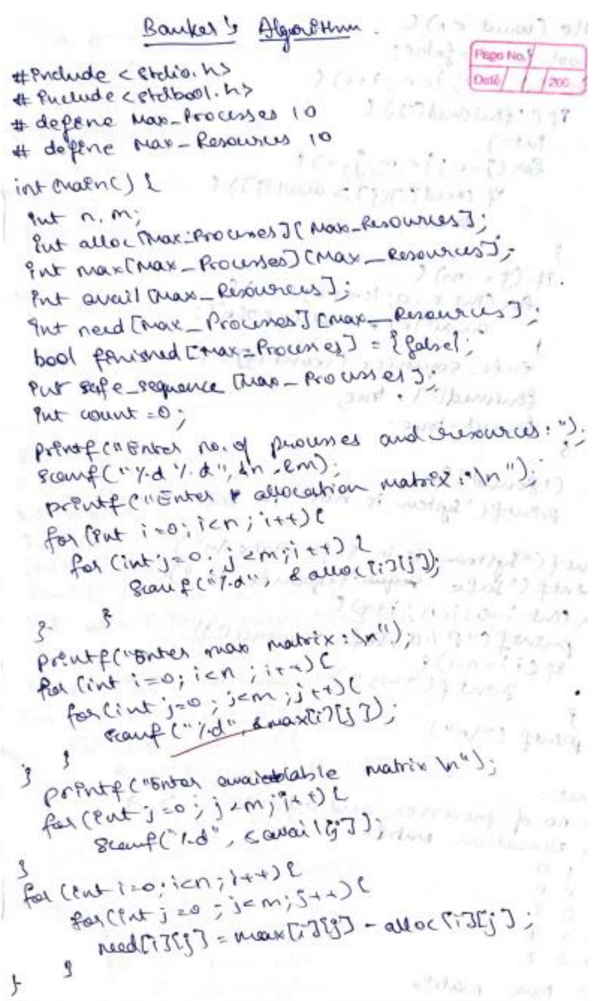
```
    finish[i] = 0;
}
for (k = 0; k < n; k++)
{
    for (i = 0; i < n; i++)
    {
        if (finish[i] == 0)
        {
            int flag = 1;
            for (j = 0; j < m; j++)
            {
                if (need[i][j] > available[j])
                {
                    flag = 0;
                    break;
                }
            }
            if (flag == 1)
            {
                safeSeq[index++] = i;
                for (j = 0; j < m; j++)
                {
                    available[j] += allocation[i][j];
                }
                finish[i] = 1;
            }
        }
    }
}
int allFinished = 1;
for (i = 0; i < n; i++)
{
    if (finish[i] == 0)
    {
        allFinished = 0;
        break;
    }
}
if (allFinished)
{
    printf("Following is the SAFE Sequence:\n");
    for (i = 0; i < n - 1; i++)
    {
        printf("P%d -> ", safeSeq[i]);
    }
```

```
        printf("P%d\n", safeSeq[n - 1]);
    }
    else
    {
        printf("The system is NOT in a safe state.\n");
    }

    return 0;
}
```

Output:

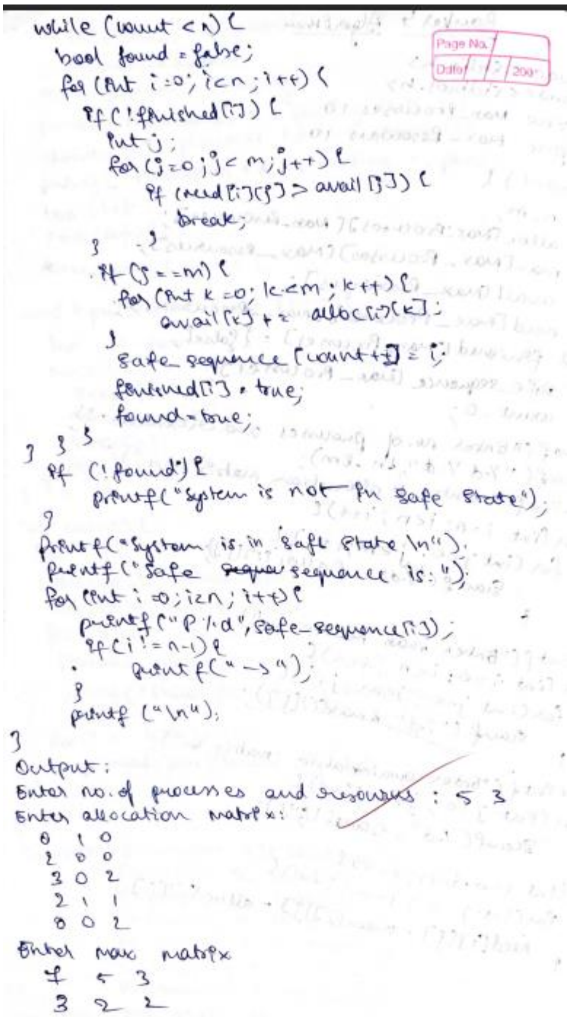


```
rs.c -o Bankers } ; if ($?) { .\Bankers }
Enter number of processes and number of resources required
5 3
Enter the max matrix for all process
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter number of allocated resources 5 for each process
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter number of available resources
3 3 2
Resouces can be allocated to Process:2 and available resources are: 3 3 2
Resouces can be allocated to Process:4 and available resources are: 5 3 2
Resouces can be allocated to Process:5 and available resources are: 7 4 3
Resouces can be allocated to Process:1 and available resources are: 7 4 5
Resouces can be allocated to Process:3 and available resources are: 7 5 5

Need Matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

System is in safe mode
<P2 P4 P5 P1 P3 >
```



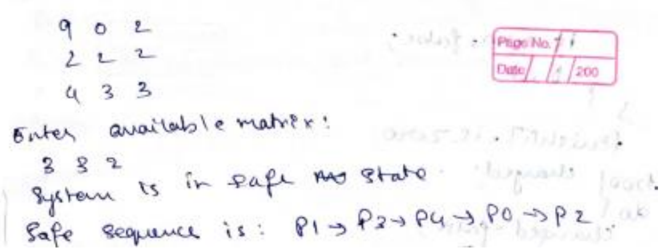
Banker's Algorithm

```
#include <stdio.h>
#include <stdbool.h>
#define Max_Processes 10
#define Max_Resources 10
int main() {
 int n, m;
 int alloc[Max_Processes][Max_Resources];
 int max[Max_Processes][Max_Resources];
 int avail[Max_Resources];
 int need[Max_Processes][Max_Resources];
 bool finished[Max_Processes] = {false};
 int safe_sequence[Max_Processes];
 int count = 0;
 printf("Enter no. of processes and resources: ");
 scanf("%d %d", &n, &m);
 printf("Enter allocation matrix:\n");
 for (int i=0; i<n; i++) {
   for (int j=0; j<m; i++) {
       scanf("%d", &alloc[i][j]);
   }
 }
 printf("Enter max matrix:\n");
 for (int i=0; i<n ; i++) {
   for (int j=0 ; j<m ; j++) {
       scanf("%d", &max[i][j]);
   }
 }
 printf("Enter available matrix \n");
 for (int j=0; j<m; j++) {
       scanf("%d", &avail[j]);
 }
 for (int i=0; i<n; i++) {
     for (int j=0 ; j<m; j++) {
       need[i][j] = max[i][j] - alloc[i][j];
     }
 }
```



```
while (count < n) {
  bool found = false;
  for (int i=0; i<n; i++) {
    if (!finished[i]) {
      int j;
      for (j=0; j<m; j++) {
          if (need[i][j] > avail[j]) {
             break;
          }
      }
      if (j==m) {
        for (int k=0; k<m ; k++) {
            avail[k] += alloc[i][k];
        }
        safe_sequence[count++] = i;
        finished[i] = true;
        found = true;
      }
    }
  }
  if (!found) {
      printf("System is not in safe state");
  }
  printf("System is in safe state\n");
  printf("Safe ~~sequ~~ sequence is: ");
  for (int i=0; i<n; i++) {
     printf("P%d", safe_sequence[i]);
     if (i != n-1) {
         printf(" -> ");
     }
  }
  printf("\n");
}
```

```
Output:
Enter no. of processes and resources : 5 3
Enter allocation matrix:
  0 1 0
  2 0 0
  3 0 2
  2 1 1
  0 0 2
Enter max matrix
  7 5 3
  3 2 2
```



```
  9 0 2
  2 2 2
  4 3 3
Enter available matrix:
  3 3 2
System is in safe ~~no~~ state
Safe sequence is: P1 -> P3 -> P4 -> P0 -> P2
```

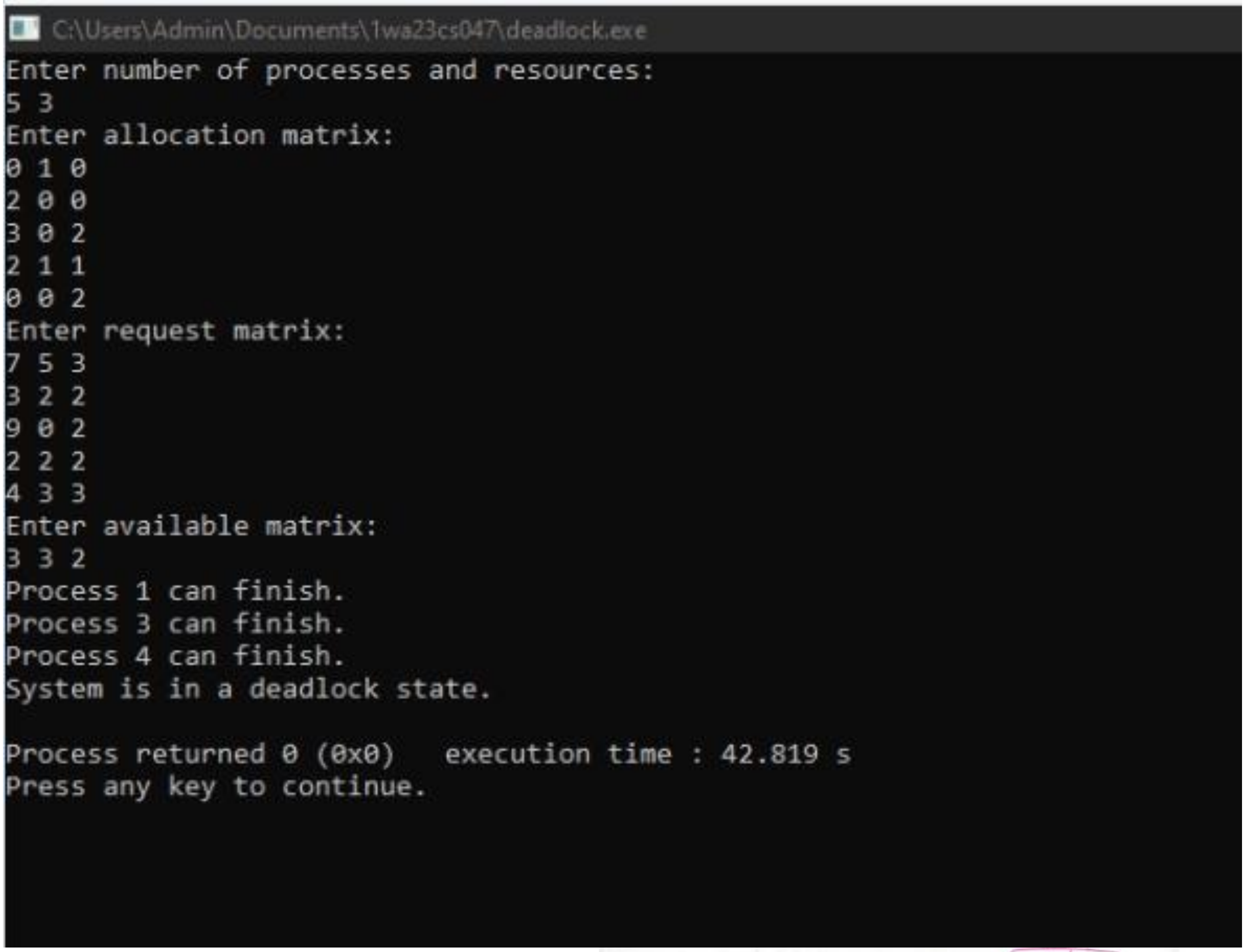
8.Write a C program to simulate deadlock detection

```
#include <stdio.h>
static int mark[20];
int i, j, np, nr;
int main()
{
    int alloc[10][10], request[10][10], avail[10], r[10], w[10];
    printf("\nEnter the number of processes: ");
    scanf("%d", &np);
    printf("\nEnter the number of resources: ");
    scanf("%d", &nr);
```

```
for (i = 0; i < nr; i++)
{
    printf("Total amount of Resource R%d: ", i + 1);
    scanf("%d", &r[i]);
}
printf("\nEnter the Request Matrix:\n");
for (i = 0; i < np; i++)
{
    for (j = 0; j < nr; j++)
    {
    scanf("%d", &request[i][j]);
    }
}
printf("\nEnter the Allocation Matrix:\n");
for (i = 0; i < np; i++)
{
    for (j = 0; j < nr; j++)
    {
        scanf("%d", &alloc[i][j]);
    }
}
for (j = 0; j < nr; j++)
{
    avail[j] = r[j];
    for (i = 0; i < np; i++)
    {
        avail[j] -= alloc[i][j];
    }
}
for (i = 0; i < np; i++)
{
    int count = 0;
    for (j = 0; j < nr; j++)
    {
        if (alloc[i][j] == 0)
            count++;
        else
            break;
    }
    if (count == nr)
        mark[i] = 1;
}
for (j = 0; j < nr; j++)
    w[j] = avail[j];
for (i = 0; i < np; i++)
```

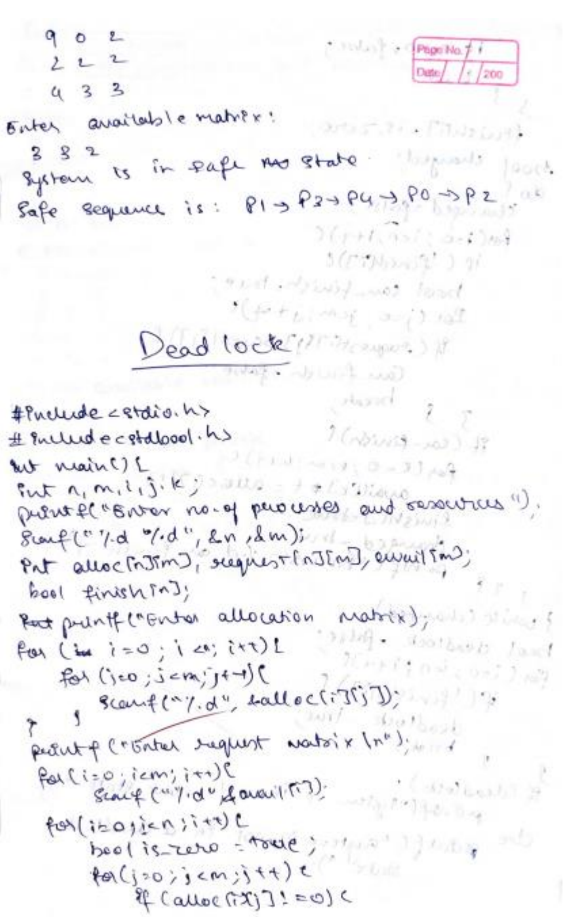
```
    {
        int canBeProcessed = 0;
        if (mark[i] != 1)
        {
            for (j = 0; j < nr; j++)
            {
                if (request[i][j] <= w[j])
                    canBeProcessed = 1;
                else {
                    canBeProcessed = 0;
                    break;
                }
            }
            if (canBeProcessed)
            {
                mark[i] = 1;
                for (j = 0; j < nr; j++)
                    w[j] += alloc[i][j];
            }
        }
    }
    int deadlock = 0;
    for (i = 0; i < np; i++)
    {
        if (mark[i] != 1)
        {
            deadlock = 1;
            break;
        }
    }

    if (deadlock)
        printf("\nDeadlock detected.\n");
    else
        printf("\nNo Deadlock possible.\n");

    return 0;
}
```

Output:



C:\Users\Admin\Documents\1wa23cs047\deadlock.exe

```
Enter number of processes and resources:
5 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter request matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter available matrix:
3 3 2
Process 1 can finish.
Process 3 can finish.
Process 4 can finish.
System is in a deadlock state.

Process returned 0 (0x0)   execution time : 42.819 s
Press any key to continue.
```

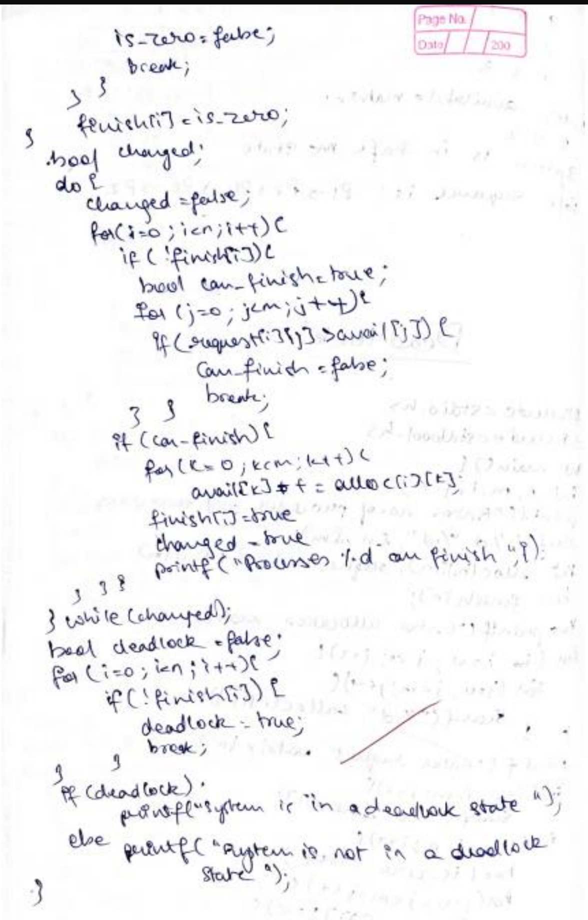


Page No. / Date / / 200

```
   9 0 2
   2 2 2
   4 3 3
Enter available matrix:
 3 3 2
System is in safe ~~no~~ state
Safe sequence is: P1 -> P3 -> P4 -> P0 -> P2
```

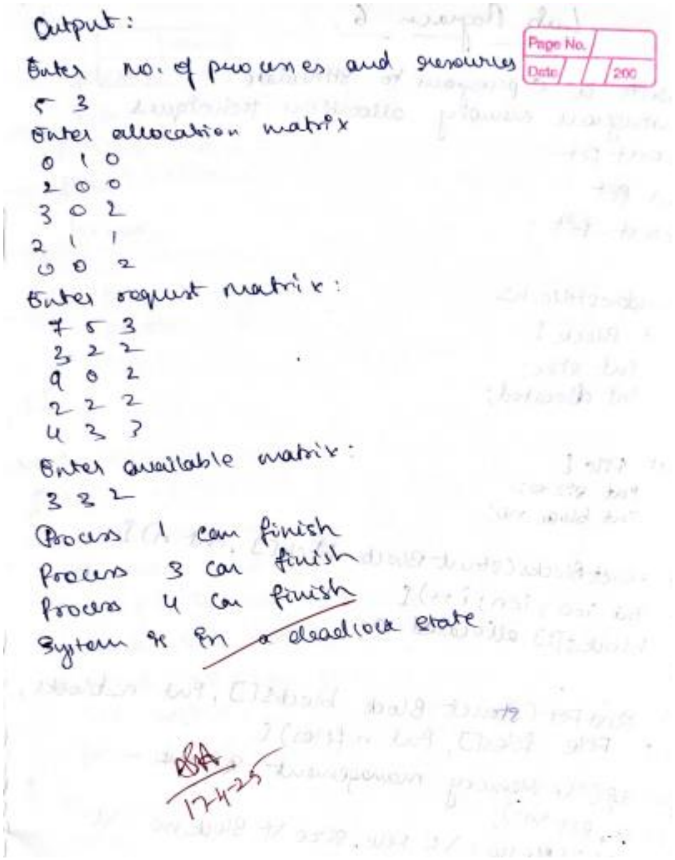
Dead lock

```
#include<stdio.h>
#include<stdbool.h>
int main(){
 int n,m,i,j,k;
 printf("Enter no.of processes and resources");
 scanf("%d %d",&n,&m);
 int alloc[n][m], request[n][m], avail[m];
 bool finish[n];
 printf("Enter allocation matrix);
 for (~~int~~ i=0; i<n; i++){
     for (j=0;j<m;j++){
         scanf("%d",&alloc[i][j]);
     }
 }
 printf("Enter request matrix \n");
 for(i=0;i<m;i++){
      scanf("%d",&avail[i]);
 for(i=0;i<n;i++){
      bool is_zero = true;
      for(j=0;j<m;j++){
          if (alloc[i][j]!=0){
```



Page No. / Date / / 200

```
             is_zero=false;
             break;
         } }
      finish[i]=is_zero;
 }
 bool changed;
 do{
   changed=false;
   for(i=0;i<n;i++){
      if (!finish[i]){
        bool can_finish=true;
        for (j=0;j<m;j++){
          if(request[i][j]>avail[j]){
              can_finish=false;
              break;
        } }
      if (can_finish){
        for(k=0;k<m;k++){
              avail[k]+=alloc[i][k];
        finish[i]=true
        changed=true
        printf("Processes %d can finish",i);
   } } }
 } while(changed);
 bool deadlock=false;
 for(i=0;i<n;i++){
      if(!finish[i]){
         deadlock=true;
         break;
      }
 }
 if(deadlock)
      printf("system is in a deadlock state");
  else printf("system is not in a deadlock
               state");
}
```



Page No. Date / / 200

```
Output:
Enter no. of processes and resources
5 3
Enter allocation matrix
 0 1 0
 2 0 0
 3 0 2
2 1 1
0 0 2
Enter request matrix:
 7 5 3
 3 2 2
 9 0 2
 2 2 2
 4 3 3
Enter available matrix:
 3 3 2
Process 1 can finish
Process 3 can finish
Process 4 can finish
System is in a deadlock state.
```

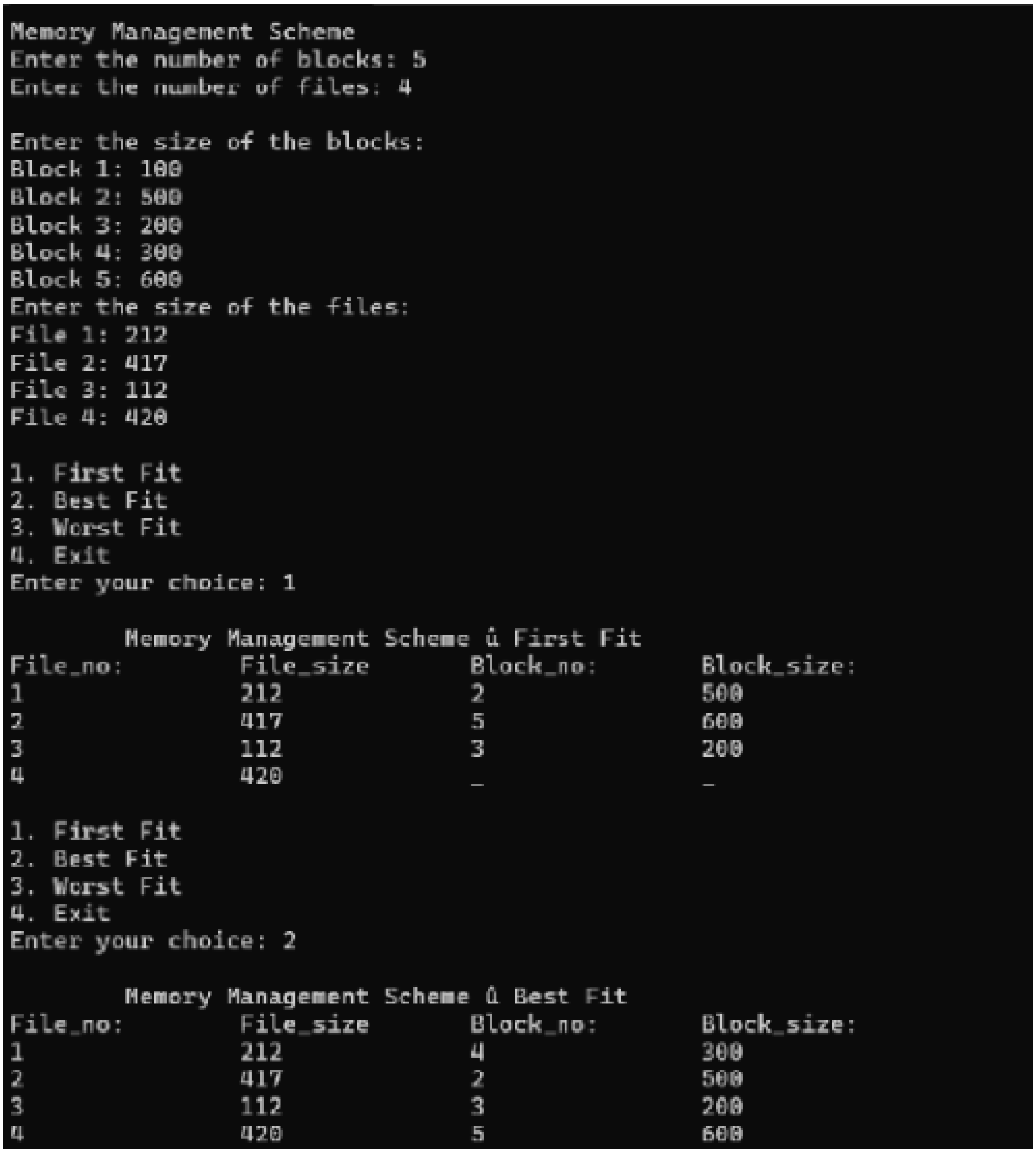
17-4-25

9.Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit
b) Best-fit
c) First-fit

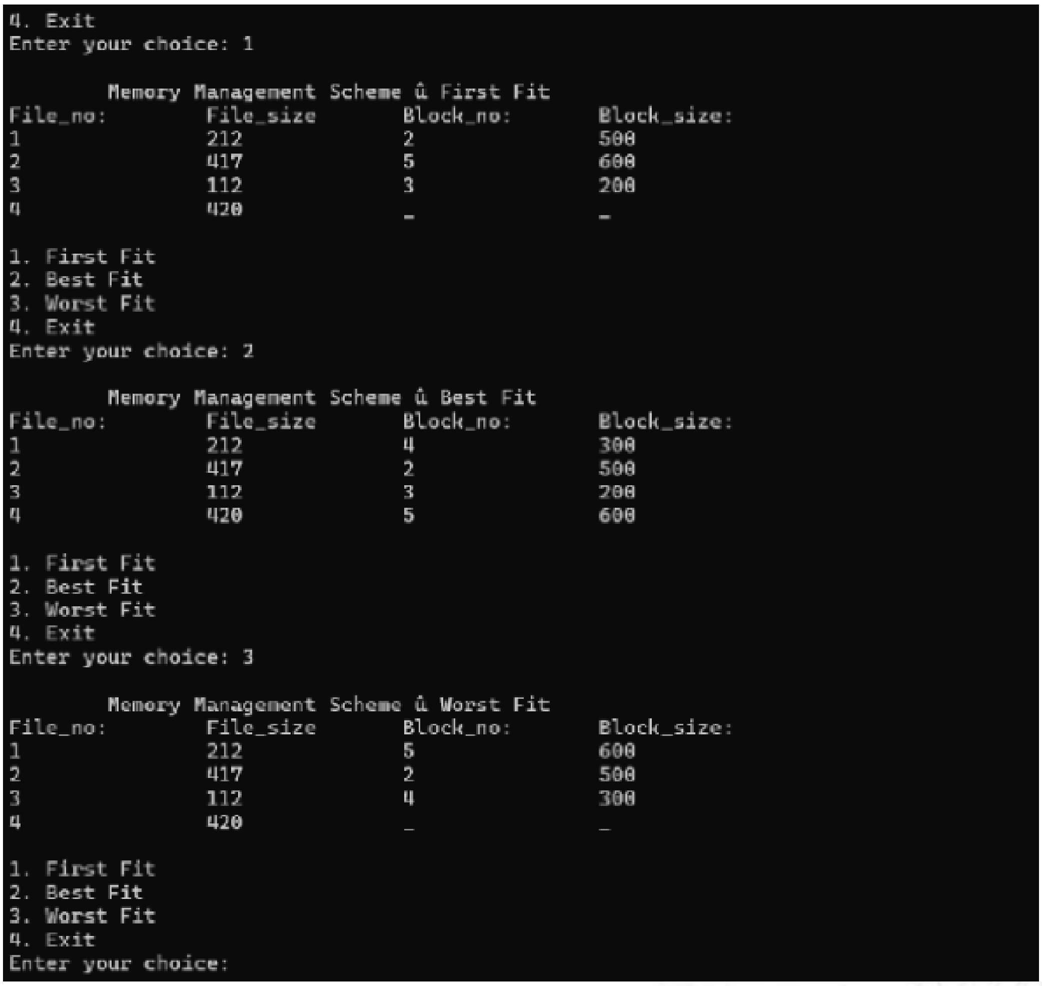
```
 #include <stdio.h>
#define MAX 25

void firstFit(int b[], int nb, int f[], int nf);
void worstFit(int b[], int nb, int f[], int nf);
void bestFit(int b[], int nb, int f[], int nf);

int main() {
    int b[MAX], f[MAX], nb, nf;
    printf("Memory Management Schemes\n");
    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 (int i = 0; i < nb; i++)
    {
        printf("Block %d: ", i + 1);
        scanf("%d", &b[i]);
    }

    printf("\nEnter the size of the files:\n");
    for (int i = 0; i < nf; i++)
    {
        printf("File %d: ", i + 1);
        scanf("%d", &f[i]);
    }

    printf("\nMemory Management Scheme - First Fit");
    firstFit(b, nb, f, nf);

    printf("\n\nMemory Management Scheme - Worst Fit");
    worstFit(b, nb, f, nf);

    printf("\n\nMemory Management Scheme - Best Fit");
    bestFit(b, nb, f, nf);

    return 0;
}

void firstFit(int b[], int nb, int f[], int nf)
{
    int bf[MAX] = {0}, ff[MAX] = {0}, frag[MAX];

    for (int i = 0; i < nf; i++)
    {
        ff[i] = -1;
        for (int j = 0; j < nb; j++)
        {
            if (!bf[j] && b[j] >= f[i])
            {
                ff[i] = j;
                bf[j] = 1;
                frag[i] = b[j] - f[i];
                break;
            }
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
    for (int i = 0; i < nf; i++)
    {
        if (ff[i] != -1)
```

```
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]);
        else
            printf("\n%d\t\t%d\t\tNot Allocated", i + 1, f[i]);
    }
}

void worstFit(int b[], int nb, int f[], int nf)
{
    int bf[MAX] = {0}, ff[MAX] = {0}, frag[MAX];

    for (int i = 0; i < nf; i++)
    {
        int worstIdx = -1;
        for (int j = 0; j < nb; j++)
        {
            if (!bf[j] && b[j] >= f[i])
            {
                if (worstIdx == -1 || b[j] - f[i] > b[worstIdx] - f[i])
                {
                    worstIdx = j;
                }
            }
        }
        ff[i] = worstIdx;
        if (worstIdx != -1)
        {
            bf[worstIdx] = 1;
            frag[i] = b[worstIdx] - f[i];
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
    for (int i = 0; i < nf; i++)
    {
        if (ff[i] != -1)
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]);
        else
            printf("\n%d\t\t%d\t\tNot Allocated", i + 1, f[i]);
    }
}

void bestFit(int b[], int nb, int f[], int nf)
{
    int bf[MAX] = {0}, ff[MAX] = {0}, frag[MAX];
    for (int i = 0; i < nf; i++)
    {
        int bestIdx = -1;
        for (int j = 0; j < nb; j++)
        {
            if (!bf[j] && b[j] >= f[i])
            {
                if (bestIdx == -1 || b[j] - f[i] < b[bestIdx] - f[i])
                {
                    bestIdx = j;
                }
            }
        }
        ff[i] = bestIdx;
        if (bestIdx != -1)
        {
            bf[bestIdx] = 1;
            frag[i] = b[bestIdx] - f[i];
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment");
    for (int i = 0; i < nf; i++)
    {
        if (ff[i] != -1)
            printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]);
        else
            printf("\n%d\t\t%d\t\tNot Allocated", i + 1, f[i]);
    }
}
```

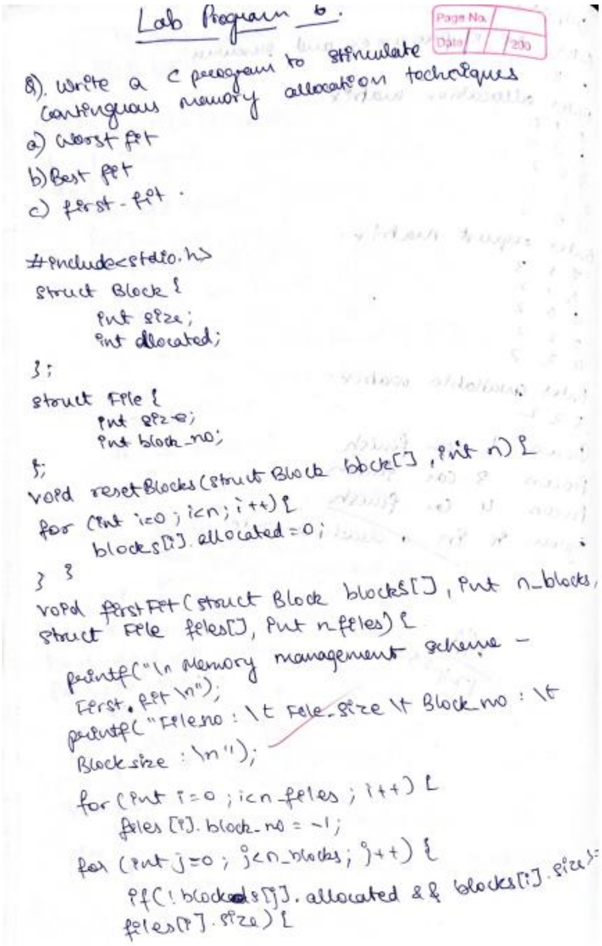
Output:



```
Memory Management Scheme
Enter the number of blocks: 5
Enter the number of files: 4

Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the size of the files:
File 1: 212
File 2: 417
File 3: 112
File 4: 420

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 1

        Memory Management Scheme û First Fit
File_no:        File_size       Block_no:       Block_size:
1               212             2               500
2               417             5               600
3               112             3               200
4               420             -               -

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2

        Memory Management Scheme û Best Fit
File_no:        File_size       Block_no:       Block_size:
1               212             4               300
2               417             2               500
3               112             3               200
4               420             5               600
```



```
4. Exit
Enter your choice: 1

        Memory Management Scheme û First Fit
File_no:        File_size       Block_no:       Block_size:
1               212             2               500
2               417             5               600
3               112             3               200
4               420             _               _

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 2

        Memory Management Scheme û Best Fit
File_no:        File_size       Block_no:       Block_size:
1               212             4               300
2               417             2               500
3               112             3               200
4               420             5               600

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice: 3

        Memory Management Scheme û Worst Fit
File_no:        File_size       Block_no:       Block_size:
1               212             5               600
2               417             2               500
3               112             4               300
4               420             _               _

1. First Fit
2. Best Fit
3. Worst Fit
4. Exit
Enter your choice:
```

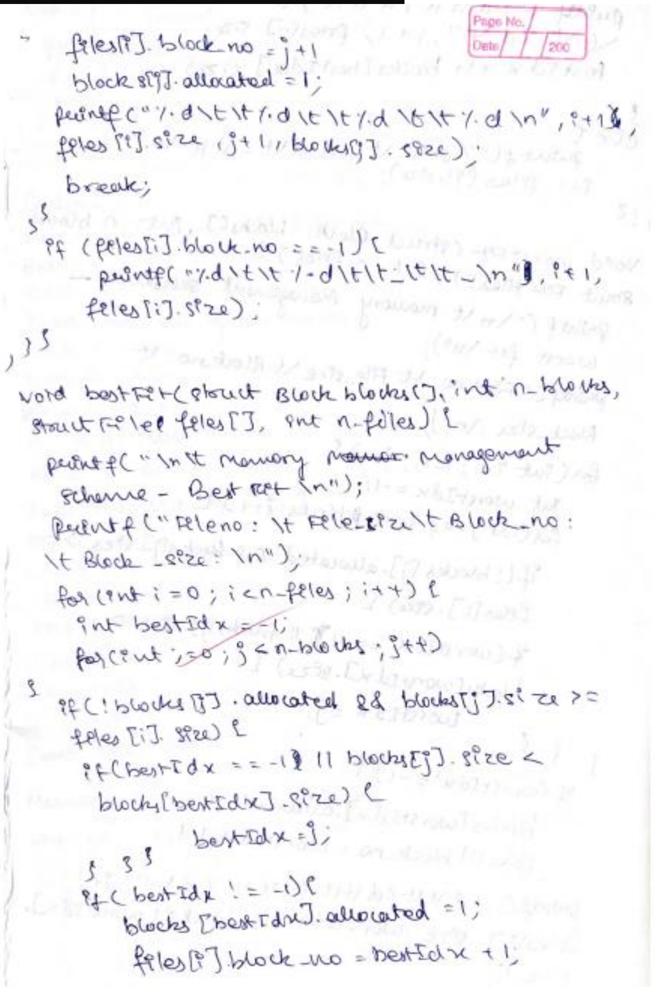


Lab Program 6.

Page No.
Date / / 200

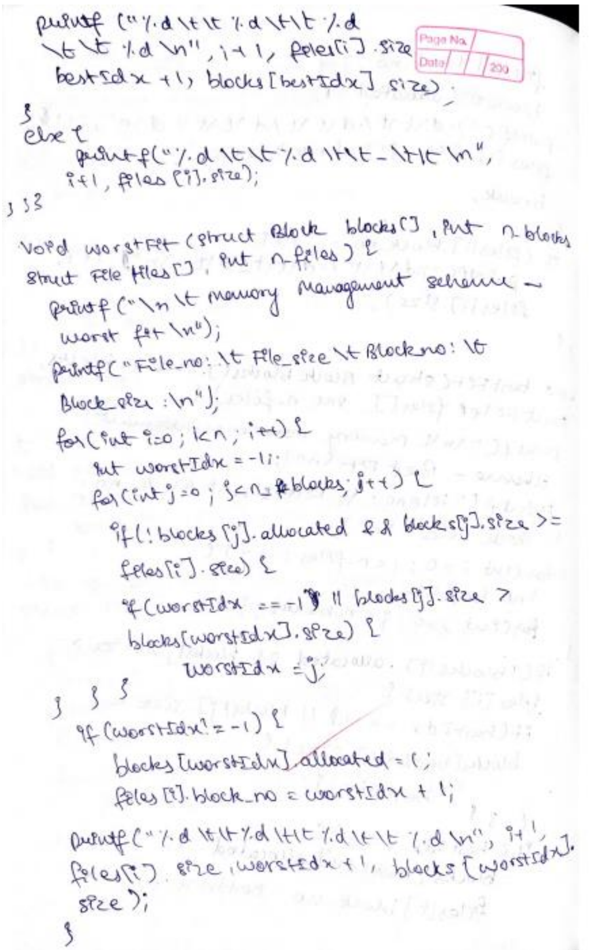
Q). Write a C program to simulate contiguous memory allocation techniques
a) worst fit
b) Best fit
c) first - fit .

```
#include<stdio.h>
 struct Block {
        int size;
        int allocated;
 };
 struct File {
        int size;
        int block_no;
 };
void resetBlocks(struct Block block[] ,int n) {
 for (int i=0 ; i<n; i++) {
        blocks[i]. allocated = 0;
 } }
void firstFit (struct Block blocks[] , int n_blocks,
struct File files[], int n_files) {
   printf("\n Memory management scheme -
   First. fit \n");
   printf("Fileno : \t File_size \t Block_no : \t
   Block size : \n");
   for (int i=0 ; i<n_files ; i++) {
        files [i]. block_no = -1;
   for (int j=0 ; j<n_blocks; j++) {
        if(!blocks[j]. allocated && blocks[i]. size>=
        files[i]. size) {
```

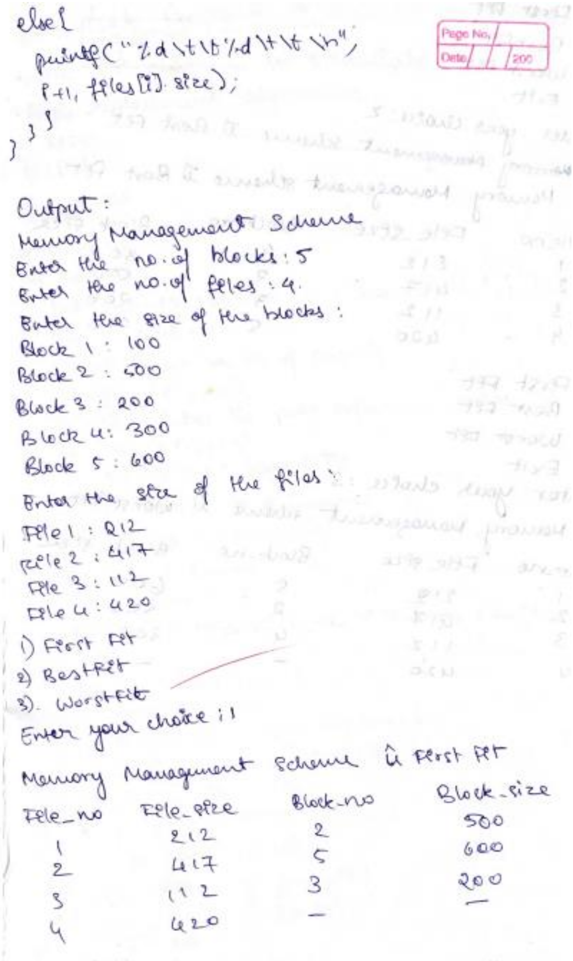


Page No.
Date / / 200

```
    files[i]. block_no = j+1
    blocks[j]. allocated = 1;
   printf("%d\t\t%d\t\t%d \t\t %d \n", i+1,
   files [i]. size ,j+1, blocks[j] . size) ;
    break;
 }}
 if (files[i].block_no == -1) {
    printf( "%d\t\t %-d\t\t_\t\t_\n", i+1,
    files[i]. size) ;
}}
void bestFit(struct Block blocks[], int n_blocks,
 struct Filel files[], int n_files) {
  printf(" \n\t memory ~~mamor~~ management
   scheme - Best fit \n");
  printf("Fileno : \t File_size\t Block_no :
 \t Block _size : \n");
  for (int i = 0 ; i<n_files ; i++) {
   int bestIdx = -1;
   for(int j=0 ; j < n_blocks ; j++)
{
  if(!blocks[j] . allocated && blocks[j].size >=
   files [i]. size) {
    if(bestIdx == -1 || blocks[j]. size <
     blocks[bestIdx].size) {
             bestIdx = j;
    } } }
   if( bestIdx != -1) {
      blocks [bestIdx]. allocated = 1;
       files[i] block_no = bestIdx + 1;
```

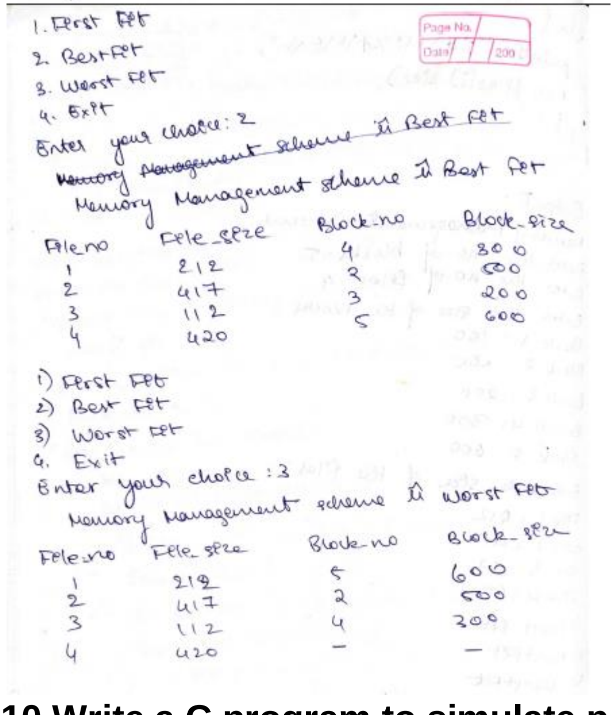


```
printf ("%d\t\t %d\t\t %d
\t\t %d \n", i+1, files[i].size
bestIdx +1, blocks[bestIdx] size);
}
else {
   printf("%d \t\t %d \t\t - \t\t \n",
  i+1, files [i].size);
} }}

void worstFit (struct Block blocks[] , int n_blocks
struct File files[] , int n_files ) {
   printf ("\n \t memory Management scheme -
   worst fit \n");
   printf("File_no: \t File_size \t Block_no: \t
   Block_size :\n");
   for(int i=0 ; i<n ; i++) {
      int worstIdx = -1;
      for(int j=0 ; j<n_blocks; j++) {
        if(!blocks[j].allocated && blocks[j].size >=
        files[i].size) {
        if(worstIdx ==-1 || blocks[j].size >
        blocks[worstIdx].size) {
              worstIdx =j;
 }  }  }
   if (worstIdx!=-1) {
      blocks [worstIdx].allocated = 1;
      files [i].block_no = worstIdx + 1;

  printf("%d \t\t %d \t\t %d \t\t %d \n", i+1,
  files[i] size ,worstIdx +1, blocks [worstIdx].
   size );
  }
```



```
else{
 printf("%d \t\t %d \t\t \n",
 i+1, files[i].size);
 } }
}
```

```
Output :
Memory Management Scheme
Enter the no.of blocks: 5
Enter the no.of files : 4.
Enter the size of the blocks :
Block 1 : 100
Block 2 : 500
Block 3 : 200
Block 4: 300
Block 5 : 600
Enter the size of the files :
File 1 : 212
File 2 : 417
File 3 : 112
File 4 : 420
1) First Fit
2) BestFit
3). WorstFit
Enter your choice :1
Memory Management Scheme is First Fit
File_no     File_size     Block_no     Block_size
  1            212            2            500
  2            417            5            600
  3            112            3            200
  4            420            -             -
```

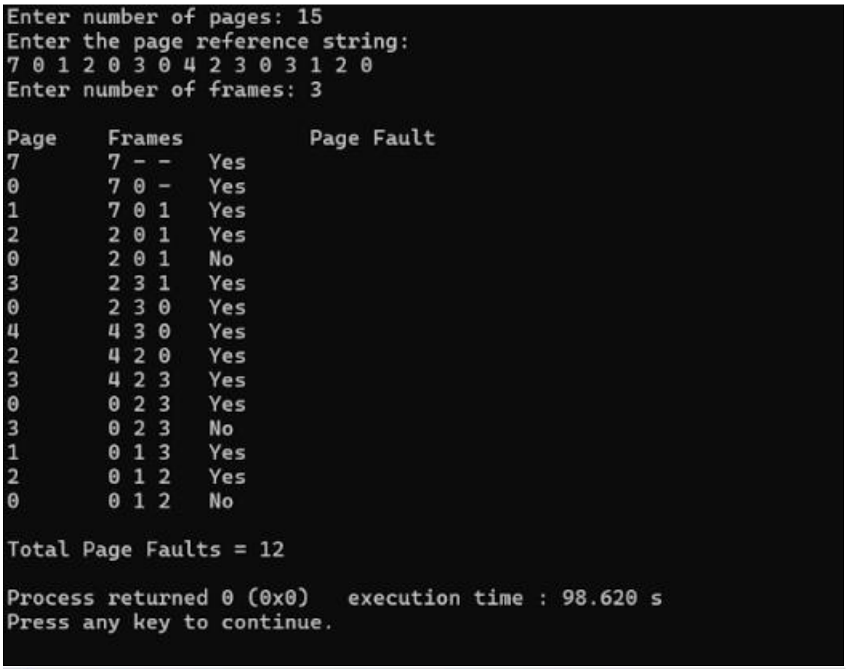


```
1. First Fit
2. BestFit
3. Worst Fit
4. Exit
Enter your choice: 2
~~Memory Management scheme is Best Fit~~
  Memory Management scheme is Best Fit
File no     File_size     Block_no     Block_size
  1            212            4            300
  2            417            2            500
  3            112            3            200
  4            420            5            600

1) First Fit
2) Best Fit
3) Worst Fit
4. Exit
Enter your choice :3
  Memory Management scheme is Worst Fit
File_no     File_size     Block_no     Block_size
  1            212            5            600
  2            417            2            500
  3            112            4            300
  4            420            -             -
```

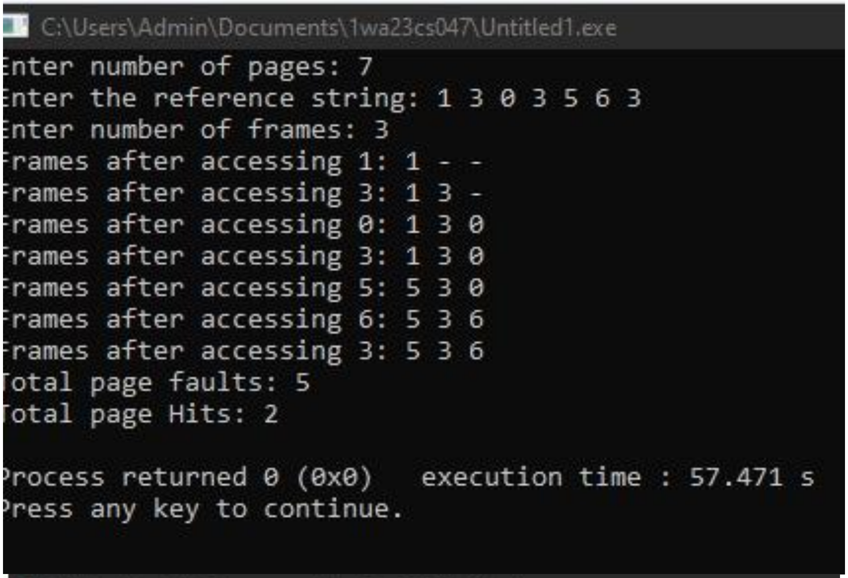
10.Write a C program to simulate page replacement algorithms LRU-Optimal-FIFO

```
#include <stdio.h>

int n, f, i, j, k;
int in[100];
int p[50];
int hit = 0;
int pgfaultcnt = 0;

void getData() {
    printf("\nEnter length of page reference sequence: ");
    scanf("%d", &n);
    printf("\nEnter the page reference sequence: ");
    for(i = 0; i < n; i++)
        scanf("%d", &in[i]);
    printf("\nEnter number of frames: ");
    scanf("%d", &f);
}

void initialize() {
    pgfaultcnt = 0;
    for(i = 0; i < f; i++)
        p[i] = 9999;
}

int isHit(int data) {
    hit = 0;
    for(j = 0; j < f; j++) {
        if(p[j] == data) {
            hit = 1;
            break;
        }
    }
    return hit;
}

void dispPages() {
    for (k = 0; k < f; k++) {
        if(p[k] != 9999)
            printf(" %d", p[k]);
    }
    printf("\n");
}

void dispPgFaultCnt() {
    printf("\nTotal number of page faults: %d\n", pgfaultcnt);
}

void fifo() {
    initialize();
    int index = 0;
    for(i = 0; i < n; i++) {
        printf("For %d :", in[i]);
        if(isHit(in[i]) == 0) {
            p[index] = in[i];
            index = (index + 1) % f;
            pgfaultcnt++;
            printf(" Page Fault ->");
            dispPages();
        } else {
            printf(" No page fault\n");
        }
    }
    dispPgFaultCnt();
}

void optimal() {
    initialize();
    int near[50];
    for(i = 0; i < n; i++) {
        printf("For %d :", in[i]);
        if(isHit(in[i]) == 0) {
            for(j = 0; j < f; j++) {
                int pg = p[j];
                int found = 0;
                for(k = i + 1; k < n; k++) {
                    if(pg == in[k]) {
                        near[j] = k;
                        found  =  1;
                        break;
                    }
                }
                if(!found)
                    near[j] = 9999;
            }
            int max = -1, repindex = -1;
            for(j = 0; j < f; j++) {
                if(near[j] > max) {
                    max = near[j];
                    repindex = j;
                }
            }
            p[repindex] = in[i];
            pgfaultcnt++;
            printf(" Page Fault ->");
            dispPages();
        } else {
            printf(" No page fault\n");
        }
    }
    dispPgFaultCnt();
}
void lru() {
    initialize();
    int least[50];
    for(i = 0; i < n; i++) {
        printf("For %d :", in[i]);
        if(isHit(in[i]) == 0) {
            for(j = 0; j < f; j++) {
                int pg = p[j];
                int found = 0;
                for(k = i - 1; k >= 0; k--) {
                    if(pg == in[k]) {
                        least[j] = k;
                        found = 1;
                        break;
                    }
                }
                if(!found)
                    least[j] = -1;
            }
            int min = 9999, repindex = -1;
            for(j = 0; j < f; j++) {
                if(least[j] < min) {
                    min = least[j];
                    repindex = j;
                }
            }
            p[repindex] = in[i];
            pgfaultcnt++;
            printf(" Page Fault ->");
            dispPages();
        } else {
            printf(" No page fault\n");
        }
    }
    dispPgFaultCnt();
}

int main() {
    int choice;
    while(1) {
        printf("\nPage Replacement Algorithms\n");
        printf("1. Enter data\n");
        printf("2. FIFO\n");
        printf("3. Optimal\n");
        printf("4. LRU\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1: getData(); break;
            case 2: fifo(); break;
            case 3: optimal(); break;
            case 4: lru(); break;
            case 5: return 0;
            default: printf("Invalid choice. Try again.\n");
        }
    }
}
```

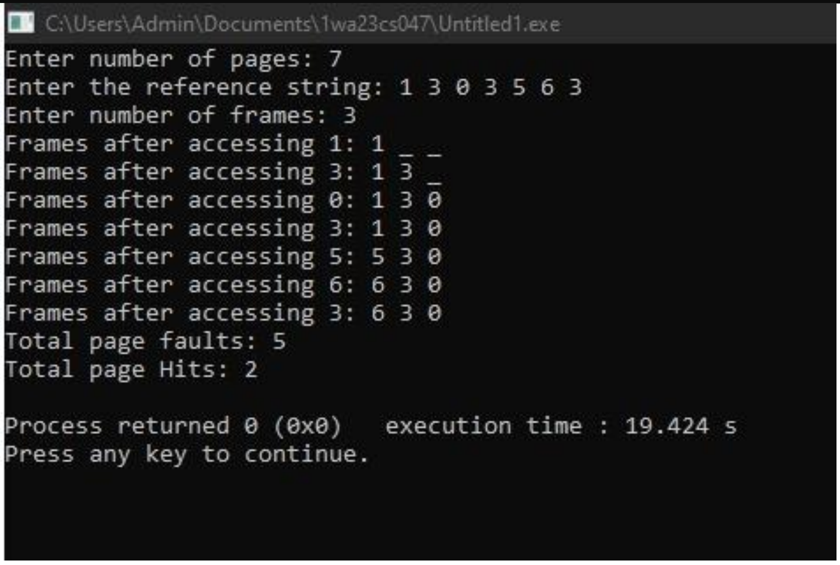
Output:



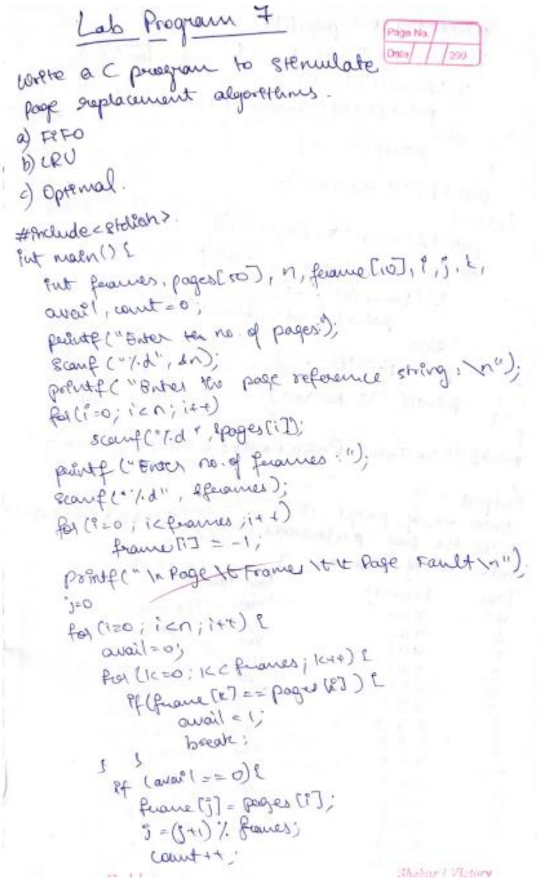
```
Enter number of pages: 15
Enter the page reference string:
7 0 1 2 0 3 0 4 2 3 0 3 1 2 0
Enter number of frames: 3

Page    Frames          Page Fault
7       7 - -   Yes
0       7 0 -   Yes
1       7 0 1   Yes
2       2 0 1   Yes
0       2 0 1   No
3       2 3 1   Yes
0       2 3 0   Yes
4       4 3 0   Yes
2       4 2 0   Yes
3       4 2 3   Yes
0       0 2 3   Yes
3       0 2 3   No
1       0 1 3   Yes
2       0 1 2   Yes
0       0 1 2   No

Total Page Faults = 12

Process returned 0 (0x0)   execution time : 98.620 s
Press any key to continue.
```



```
Enter number of pages: 7
Enter the reference string: 1 3 0 3 5 6 3
Enter number of frames: 3
Frames after accessing 1: 1 - -
Frames after accessing 3: 1 3 -
Frames after accessing 0: 1 3 0
Frames after accessing 3: 1 3 0
Frames after accessing 5: 5 3 0
Frames after accessing 6: 5 3 6
Frames after accessing 3: 5 3 6
Total page faults: 5
Total page Hits: 2

Process returned 0 (0x0)   execution time : 57.471 s
Press any key to continue.
```



```
Enter number of pages: 7
Enter the reference string: 1 3 0 3 5 6 3
Enter number of frames: 3
Frames after accessing 1: 1 _ _
Frames after accessing 3: 1 3 _
Frames after accessing 0: 1 3 0
Frames after accessing 3: 1 3 0
Frames after accessing 5: 5 3 0
Frames after accessing 6: 6 3 0
Frames after accessing 3: 6 3 0
Total page faults: 5
Total page Hits: 2

Process returned 0 (0x0)   execution time : 19.424 s
Press any key to continue.
```

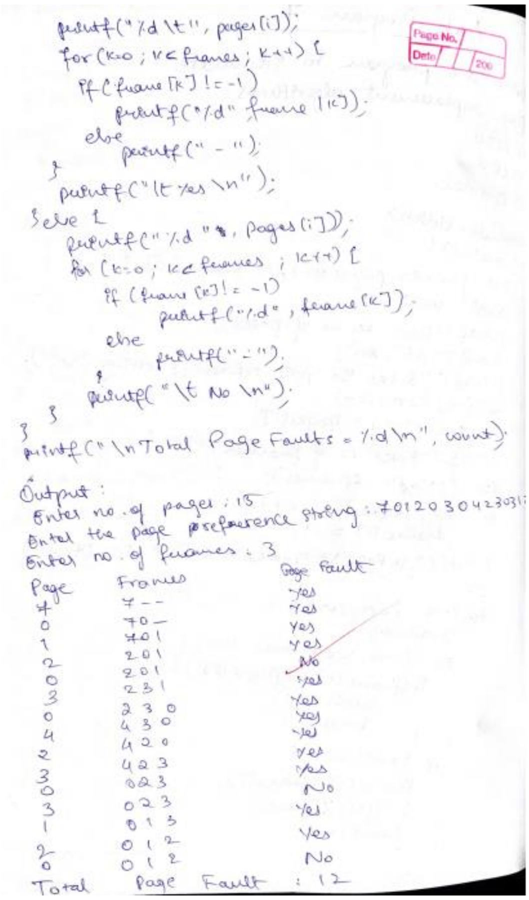


Lab Program 7

Write a C program to simulate page replacement algorithms.

a) FIFO
b) LRU
c) Optimal.

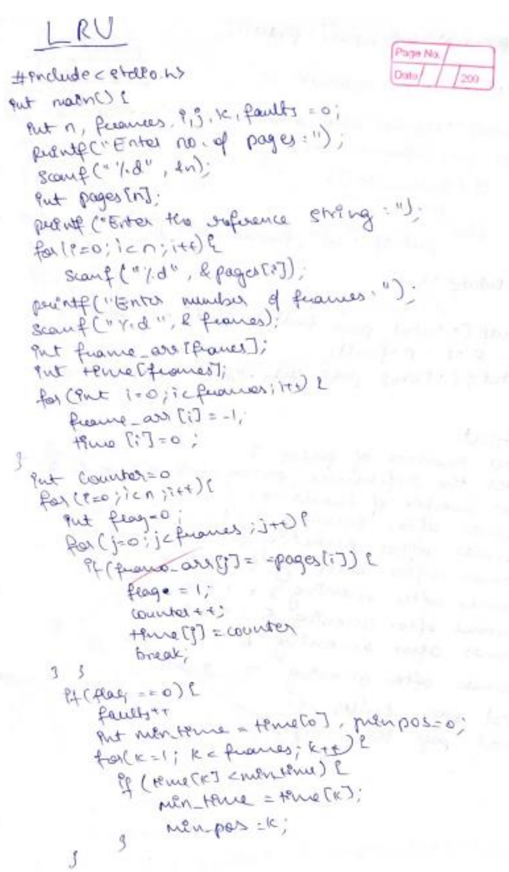
```
#include<stdio.h>
int main() {
  int frames, pages[50], n, frame[10], i, j, k,
  avail, count=0;
  printf("Enter the no. of pages");
  scanf("%d", &n);
  printf("Enter the page reference string:\n");
  for(i=0; i<n; i++)
       scanf("%d", &pages[i]);
  printf("Enter no. of frames :");
  scanf("%d", &frames);
  for(i=0; i<frames; i++)
       frame[i] = -1;
  printf("\n Page \t Frames \t\t Page Fault\n").
  j=0
  for(i=0; i<n; i++) {
     avail=0;
     for(k=0; k<frames; k++) {
        if(frame[k] == pages[i]) {
            avail = 1;
            break;
     } }
     if (avail == 0) {
        frame[j] = pages[i];
        j =(j+1) % frames;
        count++;
```



```
    printf("%d \t", pages[i]);
    for(k=0; k<frames; k++) {
      if(frame[k] != -1)
         printf("%d" frame[k]).
      else
         printf(" - ");
    }
    printf("\t Yes \n");
  }else {
    printf("%d ", pages[i]));
    for (k=0; k<frames ; k++) {
       if (frame[k]!= -1)
           printf("%d", frame[k]);
       else
           printf("-").
       }
       printf( "\t No \n").
  }
}
printf("\nTotal Page Faults = %d\n", count).
```

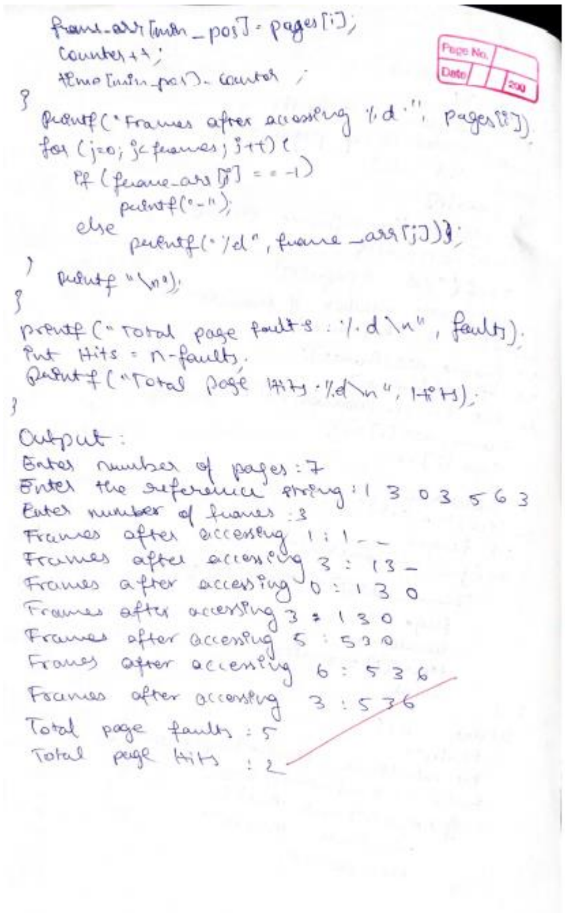
Output :

```
Enter no. of pages : 15
Enter the page preference string : 70120304230317
Enter no. of frames : 3
Page    Frames        Page Fault
                      Yes
7       7 - -         Yes
0       7 0 -         Yes
1       7 0 1         Yes
2       2 0 1         No
0       2 0 1         Yes
3       2 3 1         Yes
0       2 3 0         Yes
4       4 3 0         Yes
2       4 2 0         Yes
3       4 2 3         Yes
0       0 2 3          No
3       0 2 3         Yes
1       0 1 3         Yes
2       0 1 2         No
0       0 1 2
Total   Page  Fault  : 12
```



LRU

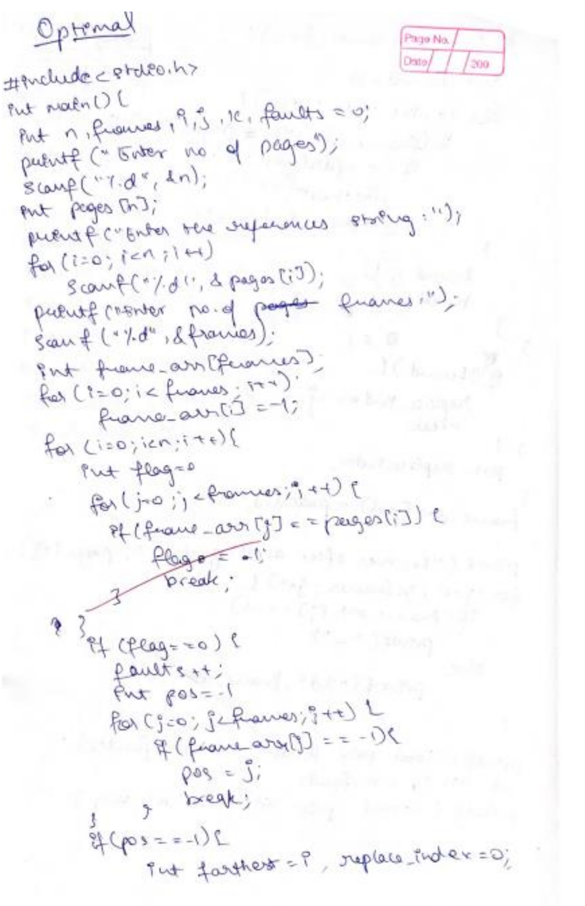
```
#include<stdio.h>
int main() {
  int n, frames, i, j, k, faults = 0;
  printf("Enter no. of pages :");
  scanf("%d", &n);
  int pages[n];
  printf("Enter the reference string :");
  for(i=0; i<n; i++){
     scanf("%d", &pages[i]);
  printf("Enter number of frames :");
  scanf("%d", & frames);
  int frame_arr[frames];
  int time[frames];
  for(int i=0; i<frames; i++) {
     frame_arr[i] = -1;
     time [i]=0 ;
}
 int counter=0
 for(i=0; i<n; i++){
   int flag=0;
   for(j=0; j<frames; j++){
     if(frame_arr[j]= =pages[i]) {
         flag = 1;
         counter++;
         time[j] = counter
         break;
   } }
   if(flag ==0) {
     faults++
     int mintime = time[0], minpos=0;
     for(k=1; k<frames; k++) {
       if (time[k] <mintime) {
           min_time = time[k];
           min_pos =k;
       }
   }
```



```
    frame_arr[min_pos] = pages[i];
    counter++;
    time[min_pos] = counter ;
 }
  printf("Frames after accessing %d :", pages[i])
  for (j=0; j<frames; j++) {
     if (frame_arr[j] = = -1)
         printf("-");
     else
         printf("%d", frame_arr[j])};
  }
  printf "\n");
}
printf("Total page faults : %d\n", faults).
int Hits = n-faults.
printf("Total page Hits : %d\n", Hits);
}
```

Output :

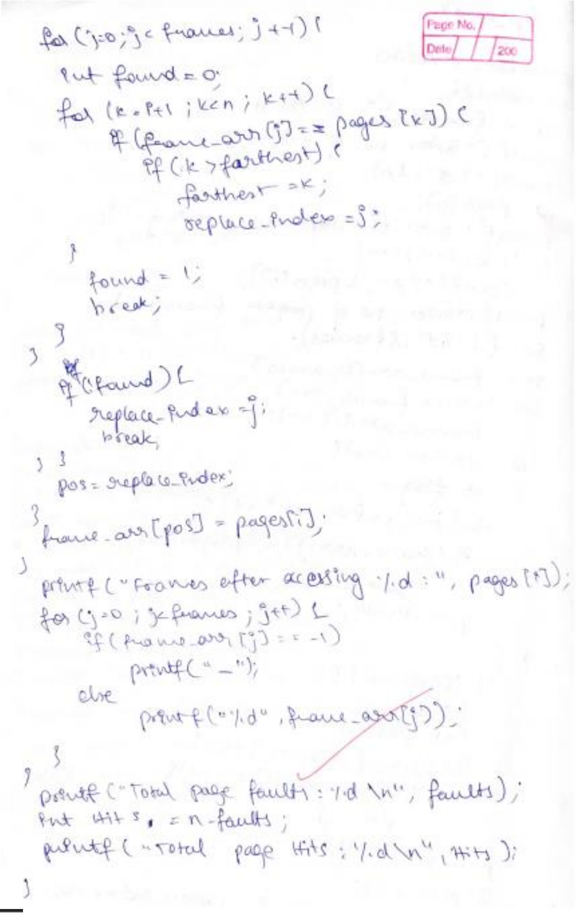
```
Enter number of pages : 7
Enter the reference string : 1 3 0 3 5 6 3
Enter number of frames : 3
Frames after accessing 1 : 1 - -
Frames after accessing 3 : 1 3 -
Frames after accessing 0 : 1 3 0
Frames after accessing 3 : 1 3 0
Frames after accessing 5 : 5 3 0
Frames after accessing 6 : 5 3 6
Frames after accessing 3 : 5 3 6
Total page faults : 5
Total page Hits : 2
```



Optimal

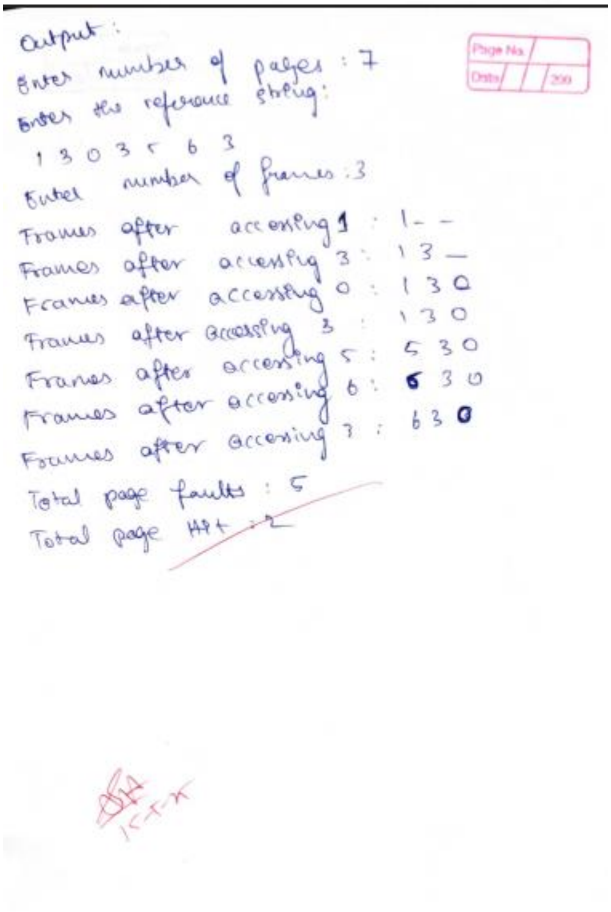
Page No. / Date / / 200

```
#include<stdio.h>
int main(){
 int n,frames,i,j,k,faults=0;
 printf("Enter no of pages");
 scanf("%d",&n);
 int pages[n];
 printf("Enter tere references string:");
 for(i=0;i<n;i++)
     scanf("%d",&pages[i]);
 printf("Enter no.of ~~pages~~ frames:");
 scanf("%d",&frames);
 int frame_arr[frames];
 for(i=0;i<frames;i++)
       frame_arr[i]=-1;
  for(i=0;i<n;i++){
     int flag=0
      for(j=0;j<frames;j++){
        if(frame_arr[j]==pages[i]){
            flag=1;
            break;
        }
   }
     if(flag==0){
       faults++;
       int pos=-1
       for(j=0;j<frames;j++){
           if(frame_arr[j]==-1){
              pos=j;
              break;
           }
     }
     if(pos==-1){
          int farthest=i, replace_index=0;
```



Page No. / Date / / 200

```
for(j=0;j<frames;j++){
  int found=0;
  for(k=i+1;k<n;k++){
      if(frame_arr[j]==pages[k]){
        if(k>farthest){
            farthest=k;
            replace_index=j;
    }
     found=1;
     break;
  }
}
  if(!found){
    replace_index=j;
    break;
} }
  pos=replace_index;
}
frame_arr[pos]=pages[i];
}
 printf("Frames after accessing %d:",pages[i]);
 for(j=0;j<frames;j++){
     if(frame_arr[j]==-1)
         printf("_");
    else
         printf("%d",frame_arr[j]);
  }
}
 printf("Total page faults:%d\n",faults);
 int Hits=n-faults;
 printf("Total page Hits:%d\n",Hits);
}
```



Output:

Page No. / Date / / 200

```
Enter number of pages : 7
Enter the reference string:
 1 3 0 3 5 6 3
Enter number of frames :3
Frames after  accessing 1 : 1 - -
Frames after  accessing 3 : 1 3 _
Frames after  accessing 0 : 1 3 0
Frames after accessing  3 : 1 3 0
Frames after  accessing 5 : 5 3 0
Frames after accessing 6 :  6 3 0
Frames after accessing 3 :  6 3 0
Total page faults : 5
Total page Hit : 2
```