

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Kamataka.



LAB REPORT

on

OPERATING SYSTEMS

Submitted by

DHRUTHI VIJAY(1BM23CS369)

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 to 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 – 23CS4PCOPS” carried out by DHRUTHI VIJAY (1BM23CS369), who is Bonafide student of B. M. S. College of Engineering. It is in partial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year Feb 2025- June 2025. The Lab report has been approved as it satisfies the academic requirements in respect of a

OPERATING SYSTEMS - (23CS4PCOPS) work prescribed for the said degree.

Ms. Sandhya A Kulkarni
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Sl. No.	Experiment Title	Page No.
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-preemptive)	1-6
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)	7-10
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.	11-13
4.	Write a C program to simulate Real-Time CPU Scheduling algorithms: a) Rate- Monotonic b) Earliest-deadline First	14-18
5.	Write a C program to simulate producer-consumer problem using semaphores	19-20
6.	Write a C program to simulate the concept of Dining Philosophers problem.	21-23
7.	Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.	24-25
8.	Write a C program to simulate deadlock detection	26-28
9.	Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b) Best-fit c) First-fit	29-33
10.	Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) OPTIMAL	34-39

Course Outcomes

C01	Apply the different concepts and functionalities of Operating System
C02	Analyze various Operating system strategies and techniques
C03	Demonstrate the different functionalities of Operating System.
C04	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-preemptive)

=>FCFS:

```
#include <stdio.h>
void calculateTimes(int processes[], int n, int at[], int bt[], int ct[], int tat[], int wt[]) {
    int completion = 0;
    for (int i = 0; i < n; i++) {
        if (completion < at[i]) {
            completion = at[i];
        }
        completion += bt[i];
        ct[i] = completion;
        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt[i];
    }
}
void displayResults(int processes[], int n, int at[], int bt[], int ct[], int tat[], int wt[]) {
    float total_tat = 0, total_wt = 0;
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        total_tat += tat[i];
        total_wt += wt[i];
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", processes[i], at[i], bt[i], ct[i], tat[i], wt[i]);
    }
    printf("\nAverage Turnaround Time = %.2f", total_tat / n);
    printf("\nAverage Waiting Time = %.2f\n", total_wt / n);
}
int main() {
    int n;
    printf("Enter total number of processes: ");
    scanf("%d", &n);

    int processes[n], at[n], bt[n];
    for (int i = 0; i < n; i++) {
        printf("Enter process number: ");
        scanf("%d", &processes[i]);
        printf("Enter arrival time: ");
        scanf("%d", &at[i]);
        printf("Enter burst time: ");
        scanf("%d", &bt[i]);
    }
}
```

```

int ct[n], tat[n], wt[n];
calculateTimes(processes, n, at, bt, ct, tat, wt);
displayResults(processes, n, at, bt, ct, tat, wt);

return 0;
}

```

RESULT:

```

PS C:\Users\Admin\Documents\OSLAB> cd "c:\Users\Admin\Documents\OSLAB\" ; if ($?) { gcc fcfs.c -o fcfs } ; if ($?) { ./fcfs }

enter total number of processes: 4
enter process number: 1
enter arrival time: 0
enter burst time: 5
enter process number: 2
enter arrival time: 1
enter burst time: 3
enter process number: 3
enter arrival time: 2
enter burst time: 8
enter process number: 4
enter arrival time: 3
enter burst time: 6

Process AT BT CT TAT WT
1 0 5 5 5 0
2 1 3 8 7 4
3 2 8 16 14 6
4 3 6 22 19 13

Average Turnaround Time = 11.25
Average Waiting Time = 5.75

```

KRUPA					
Date	Page				
06/07/2025					
<u>LAB-1</u>					
Write a C program to simulate the following non-preemptive and preemptive CPU scheduling algorithms to find their turnaround time and waiting time:					
<ul style="list-style-type: none"> • FCFS • SJF (Non-preemptive & preemptive) 					
//FCFS					
<pre> void calculateTimes(int processes[], int n, int at[], int bt[], int bt[], int ct[], int tat[], int wt[]) { int completion = 0; for (int i=0; i<n; i++) { if (completion < at[i]) completion = at[i]; completion += bt[i]; ct[i] = completion; tat[i] = ct[i] - at[i]; wt[i] = tat[i] - bt[i]; } } void display(int processes[], int n, int at[], int bt[], int ct[], int tat[], int wt[]) { float total_tat = 0, total_wt = 0; printf("\nProcesses AT BT CT TAT WT \n"); for (int i=0; i<n; i++) { printf("%d %d %d %d %d %d \n", i+1, at[i], bt[i], ct[i], tat[i], wt[i]); total_tat += tat[i]; total_wt += wt[i]; } printf("Average TAT = %.2f", total_tat/n); printf("Average WT = %.2f", total_wt/n); } </pre>					
for (int i=0; i<n; i++) { total_tat += tat[i]; total_wt += wt[i]; printf("\nProcesses AT BT CT TAT WT \n"); ct[i], tat[i], wt[i]); } printf("Average TAT = %.2f", total_tat/n); printf("Average WT = %.2f", total_wt/n);					
Output:					
Enter total nof processes :4 Enter process number: 1 Enter arrival time: 0 Enter burst time: 5 Enter process number: 2 Enter arrival time: 1 Enter burst time: 3 Enter process number: 3 Enter arrival time: 2 Enter burst time: 8 Enter process number: 4 Enter arrival time: 3 Enter burst time: 6					

KRUPA																																			
Date	Page																																		
06/07/2025																																			
<table border="1"> <thead> <tr> <th>Process</th> <th>AT</th> <th>BT</th> <th>CT</th> <th>TAT</th> <th>WT</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>0</td> <td>5</td> <td>5</td> <td>5</td> <td>0</td> </tr> <tr> <td>2</td> <td>1</td> <td>3</td> <td>8</td> <td>7</td> <td>4</td> </tr> <tr> <td>3</td> <td>2</td> <td>8</td> <td>16</td> <td>14</td> <td>6</td> </tr> <tr> <td>4</td> <td>3</td> <td>6</td> <td>22</td> <td>19</td> <td>13</td> </tr> </tbody> </table>						Process	AT	BT	CT	TAT	WT	1	0	5	5	5	0	2	1	3	8	7	4	3	2	8	16	14	6	4	3	6	22	19	13
Process	AT	BT	CT	TAT	WT																														
1	0	5	5	5	0																														
2	1	3	8	7	4																														
3	2	8	16	14	6																														
4	3	6	22	19	13																														
Average TAT = 11.25 Average WT = 5.75																																			
<table border="1"> <tr> <td>1</td> <td>2</td> <td>3</td> <td>4</td> </tr> <tr> <td>0</td> <td>5</td> <td>8</td> <td>16</td> <td>22</td> </tr> </table>						1	2	3	4	0	5	8	16	22																					
1	2	3	4																																
0	5	8	16	22																															

=>SJF(Non-preemptive):

```
#include <stdio.h>
#include <limits.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_SJF_NonPreemptive(struct process p[], int n) {
    int completed = 0, currentTime = 0;
    while (completed < n) {
        int shortest = -1, minBT = INT_MAX;
        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].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++;
        }
    }
}

void display(struct process p[], int n) {
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\tRT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
               p[i].id, p[i].AT, p[i].BT, p[i].CT, p[i].TAT, p[i].WT, p[i].RT);
    }
}
```

```

    }

}

int main() {
    struct process p[MAX];
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter Arrival Time (AT) for process %d: ", i + 1);
        scanf("%d", &p[i].AT);
        printf("Enter Burst Time (BT) for process %d: ", i + 1);
        scanf("%d", &p[i].BT);
        p[i].completed = 0;
    }
    calculate_SJF_NonPreemptive(p, n);
    display(p, n);

    return 0;
}

```

RESULT:

```

Enter number of processes: 4
Enter Arrival Time (AT) for process 1: 0
Enter Burst Time (BT) for process 1: 7
Enter Arrival Time (AT) for process 2: 0
Enter Burst Time (BT) for process 2: 3
Enter Arrival Time (AT) for process 3: 0
Enter Burst Time (BT) for process 3: 4
Enter Arrival Time (AT) for process 4: 0
Enter Burst Time (BT) for process 4: 6

Process AT      BT      CT      TAT      WT      RT
1      0       7       20      20      13      13
2      0       3       3       3       0       0
3      0       4       7       7       3       3
4      0       6      13      13      7       7

Process returned 0 (0x0)   execution time : 36.770 s
Press any key to continue.

```

Process	AT	BT	CT	TAT	WT	RT
1	0	7	20	20	13	13
2	0	3	3	3	0	0
3	0	4	7	7	3	3
4	0	6	13	13	7	7

Handwritten Notes:

- //non-preemptive**
- struct process {**
- int id, AT, BT, CT, TA, TAT, WT, RT, remaining_BT;**
- int completed;**
- }**
- };**
- void calculate_SJF_NonPreemptive(struct process p,**
- int n){**
- int completed = 0, currentTime = 0;**
- while(completed < n){**
- int shortest = 1, minBT = INT_MAX;**
- for(int i=0; i<n; i++){**
- if(p[i].completed == 0 & p[i].AT <= currentTime & p[i].BT < minBT)**
- minBT = p[i].BT;**
- shortest = i;**
- }**
- p[shortest].CT = currentTime + p[shortest].BT;**
- p[shortest].TAT = p[shortest].CT - p[shortest].AT;**
- p[shortest].WT = p[shortest].CT - p[shortest].AT - p[shortest].BT;**
- p[shortest].completed = 1;**
- completed++;**
- }**
- void display(struct process p[], int n){**
- for(int i=0; i<n; i++){**
- printf("%d %d %d %d %d %d %d\n",**
- p[i].id, p[i].AT, p[i].BT, p[i].CT, p[i].TAT,**
- p[i].WT, p[i].RT);**
- }**
- }**
- Output:**
- Enter number of processes: 4**
- Enter AT for process 1: 0**
- Enter BT for process 1: 7**

Handwritten Results:

Date: _____ Page: _____

if (shortest == 1) {
 currentTime++;
 p[shortest].CT = currentTime + p[shortest].BT;
 p[shortest].TAT = p[shortest].CT - p[shortest].AT;
 p[shortest].WT = p[shortest].CT - p[shortest].AT - p[shortest].BT;
 p[shortest].completed = 1;
 completed++;
}

Process AT BT CT TAT WT RT

1 0 7 20 20 13 13
2 0 3 3 3 0 0
3 0 4 7 7 3 3
4 0 6 13 13 7 7

Avg AT : 10.75
Avg WT : 5.75

=>SJF(Preemptive):

```
#define MAX 1
struct process {
    int id, AT, BT, CT, TAT, WT, RT, remaining_BT;
    int completed;
};

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 = INT_MAX;
        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++;
            }
        }
    }
}

void display(struct process p[], int n) {
    float total_tat = 0, total_wt = 0;
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\tRT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
               p[i].id, p[i].AT, p[i].BT, p[i].CT, p[i].TAT, p[i].WT, p[i].RT);
        total_tat += p[i].TAT;
        total_wt += p[i].WT;
    }
    printf("\nAvg TAT = %.2f", total_tat / n);
    printf("\nAvg WT = %.2f\n", total_wt / n);
}
```

```

int main() {
    struct process p[MAX];
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter Arrival Time (AT) for process %d: ", i + 1);
        scanf("%d", &p[i].AT);
        printf("Enter Burst Time (BT) for process %d: ", i + 1);
        scanf("%d", &p[i].BT);
        p[i].completed = 0;
    }
    calculate_SJF_Preemptive(p, n);
    display(p, n);

    return 0;
}

```

Result:

```

Enter number of processes: 4
Enter Arrival Time (AT) for process 1: 0
Enter Burst Time (BT) for process 1: 8
Enter Arrival Time (AT) for process 2: 1
Enter Burst Time (BT) for process 2: 4
Enter Arrival Time (AT) for process 3: 2
Enter Burst Time (BT) for process 3: 9
Enter Arrival Time (AT) for process 4: 3
Enter Burst Time (BT) for process 4: 5

Process AT BT CT TAT WT RT
1 0 8 17 17 9 0
2 1 4 5 4 0 0
3 2 9 26 24 15 15
4 3 5 10 7 2 2

avg tat= 13.00
avg wt= 6.50
Process returned 0 (0x0) execution time : 11.644 s
Press any key to continue.

```

//preemptive

```

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 = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (p[i].remaining_BT < minBT)

```

KRUPA

```

if (p[i].completed && p[i].AT <= currentTime) {
    minBT = p[i].remaining_BT;
    shortest = i;
}

if (shortest == -1) {
    currentTime++;
    select();
}

if (p[shortest].remaining_BT == p[shortest].BT) {
    p[shortest].RT = currentTime;
    p[shortest].TAT = p[shortest].AT;
    p[shortest].WT = 0;
}

p[shortest].remaining_BT -= currentTime;
if (p[shortest].remaining_BT == 0) {
    p[shortest].completed = 1;
    completed++;
}

p[shortest].CT = currentTime;
p[shortest].TAT = p[shortest].CT - p[shortest].AT;
p[shortest].WT = (shortest.TAT - p[shortest].BT);
p[shortest].completed = 1;
completed++;

void display - same as previous

```

Program -2

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

→ Priority (pre-emptive & Non-preemptive)

=>CPU SCHEDULING:

```
#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 lowest_priority = 9999, selected = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && !p[i].is_completed && p[i].pt < lowest_priority) {
                lowest_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 lowest_priority = 9999, selected = -1;
```

```

        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].remaining_bt > 0 && p[i].pt < lowest_priority) {
                lowest_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;
        }
        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%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 (lower number means higher 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;
}

```

RESULT:

```

Enter the number of processes: 4
Enter Arrival Time, Burst Time, and Priority for Process 1:
Arrival Time: 0
Burst Time: 5
Priority (lower number means higher priority): 4

Enter Arrival Time, Burst Time, and Priority for Process 2:
Arrival Time: 2
Burst Time: 4
Priority (lower number means higher priority): 2

Enter Arrival Time, Burst Time, and Priority for Process 3:
Arrival Time: 2
Burst Time: 2
Priority (lower number means higher priority): 6

Enter Arrival Time, Burst Time, and Priority for Process 4:
Arrival Time: 4
Burst Time: 4
Priority (lower number means higher priority): 3

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

PID    AT      BT      Priority       CT      TAT      WT      RT
1      0       5       4             13     13       8       0
2      2       4       2             6      4       0       0
3      2       2       6             15     13       11      11
4      4       4       3             10     6       2       2

Average TAT: 9.00
Average WT: 5.25
Average RT: 3.25

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       5       4             5      5       0       0
2      2       4       2             9      7       3       0
3      2       2       6             15     13       11      11
4      4       4       3             13     9       5       2

Average TAT: 8.50
Average WT: 4.75
Average RT: 3.25

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

```

LAB-2 : Priority Scheduling

```

#include <cs50.h>
#define MAX 10
typedef struct {
    int pid, at, bt, remaining_bt, ct, rt;
    int wt, nt, is_completed, etc;
} process;
void nonPreemptivePriority (process p[], int n) {
    int time = 0, completed = 0;
    while (completed < n) {
        int lowest_priority = 9999, selected = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].bt <= 0) {
                if (p[i].bt <= p[selected].bt) {
                    if (p[i].bt <= p[selected].bt && p[i].priority < p[selected].priority) {
                        selected = i;
                    }
                    lowest_priority = p[i].priority;
                }
            }
        }
        if (selected == -1) {
            time++;
            continue;
        }
        if (p[selected].bt == -1) {
            time++;
            continue;
        }
        if (p[selected].bt == time) {
            p[selected].ct = time;
            p[selected].remaining_bt -= time - p[selected].at;
            time += p[selected].bt;
            p[selected].ct += time;
            p[selected].wt += p[selected].at - p[selected].ct;
            p[selected].rt += p[selected].bt - (p[selected].at - p[selected].ct);
            p[selected].is_completed = 1;
        }
    }
}

```

KRUPA

```

p[Selected].wt = p[Selected].at - p[Selected].ct;
Completed += 1;
}

void preemptivePriority (process p[], int n) {
    int time = 0, completed = 0;
    while (Completed < n) {
        int lowest_priority = 9999, selected = -1;
        for (int i = 0; i < n; i++) {
            if (p[i].at <= time && p[i].bt <= 0) {
                if (p[i].bt <= p[selected].bt && p[i].priority < p[selected].priority) {
                    selected = i;
                    lowest_priority = p[i].priority;
                }
            }
        }
        if (selected == -1) {
            time++;
            continue;
        }
        if (p[selected].bt == -1) {
            time++;
            continue;
        }
        if (p[selected].bt == time) {
            p[selected].ct = time;
            p[selected].remaining_bt -= time - p[selected].at;
            time += p[selected].bt;
            p[selected].ct += time;
            p[selected].wt += p[selected].at - p[selected].ct;
            p[selected].rt += p[selected].bt - (p[selected].at - p[selected].ct);
            p[selected].is_completed = 1;
        }
    }
}

void displayProcess (process p[], int n) {
    float avg_tat = 0, avg_wt = 0, avg_rt = 0;
    printf ("PID AT BT Priority CT TAT WT RT\n");
    for (int i = 0; i < n; i++) {
        printf ("%d %d %d %d %d %d %d %d\n", p[i].pid, p[i].at, p[i].bt, p[i].priority, 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 ("\nAvg TAT: %.2f\n", avg_tat / n);
    printf ("Avg WT: %.2f\n", avg_wt / n);
    printf ("Avg RT: %.2f\n", avg_rt / n);
}

Priority scheduling Menu:
1. Non-Preemptive
2. Preemptive
3. Exit
Enter choice: 2
Preemptive scheduling completed!

PID AT BT Priority CT TAT WT RT
1 0 5 4 5 5 0 0
2 2 4 2 9 7 3 0
3 2 2 6 15 13 11 11
4 4 4 3 10 6 2 2

Avg TAT: 9.00
Avg WT: 5.25
Avg RT: 3.25
P1 P2 P3 P4 P5

Priority scheduling Menu:
1. Non-Preemptive
2. Preemptive
3. Exit
Enter choice: 1
Non-Preemptive scheduling completed!
PID AT BT Priority CT TAT WT RT
1 0 5 4 5 5 0 0
2 2 4 2 9 7 3 0
3 2 2 6 15 13 11 11
4 4 4 3 13 9 5 2

Avg TAT: 8.50
Avg WT: 4.75
Avg RT: 3.25
P1 P2 P4 P3

```

Program - 3

Write a C program to simulate a 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.

=>MULTI LEVEL SCHEDULING

```
#include <stdio.h>
struct Process {
    int id, burst_time, arrival_time, queue;
    int waiting_time, turnaround_time, response_time;
};

void round_robin(struct Process p[], int n, int quantum) {
    int remaining_time[n], completed = 0, time = 0;
    for (int i = 0; i < n; i++) remaining_time[i] = p[i].burst_time;
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (remaining_time[i] > 0) {
                if (remaining_time[i] > quantum) {
                    time += quantum;
                    remaining_time[i] -= quantum;
                } else {
                    time += remaining_time[i];
                    p[i].waiting_time = time - p[i].arrival_time - p[i].burst_time;
                    p[i].turnaround_time = time - p[i].arrival_time;
                    p[i].response_time = p[i].waiting_time;
                    remaining_time[i] = 0;
                    completed++;
                }
            }
        }
    }
}

void fcfs(struct Process p[], int n, int start_time) {
    int time = start_time;
    for (int i = 0; i < n; i++) {
        if (time < p[i].arrival_time)
            time = p[i].arrival_time;

        p[i].waiting_time = time - p[i].arrival_time;
        p[i].turnaround_time = p[i].waiting_time + p[i].burst_time;
        p[i].response_time = p[i].waiting_time;
        time += p[i].burst_time;
    }
}

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

```

struct Process processes[n], system_queue[n], user_queue[n];
int sys_count = 0, user_count = 0;

printf("Enter Burst Time, Arrival Time and Queue of each process: \n");
for (int i = 0; i < n; i++) {
    printf("P%d: ", i + 1);
    scanf("%d%d %d", &processes[i].burst_time, &processes[i].arrival_time, &processes[i].queue);
    processes[i].id = i + 1;

    if (processes[i].queue == 1)
        system_queue[sys_count++] = processes[i];
    else if (processes[i].queue == 2)
        user_queue[user_count++] = processes[i];
}

int quantum = 2;
round_robin(system_queue, sys_count, quantum);
int last_exec_time = (sys_count > 0) ? system_queue[sys_count - 1].turnaround_time : 0;
fcfs(user_queue, user_count, last_exec_time);

printf("\nProcess\tWaiting Time\tTurn Around Time\tResponse Time\n");
for (int i = 0; i < sys_count; i++)
    printf("P%d\t%d\t%d\t%d\n", system_queue[i].id, system_queue[i].waiting_time,
           system_queue[i].turnaround_time, system_queue[i].response_time);

for (int i = 0; i < user_count; i++)
    printf("P%d\t%d\t%d\t%d\n", user_queue[i].id, user_queue[i].waiting_time,
           user_queue[i].turnaround_time, user_queue[i].response_time);

float avg_wait = 0, avg_tat = 0, avg_resp = 0;
for (int i = 0; i < sys_count; i++) {
    avg_wait += system_queue[i].waiting_time;
    avg_tat += system_queue[i].turnaround_time;
    avg_resp += system_queue[i].response_time;
}
for (int i = 0; i < user_count; i++) {
    avg_wait += user_queue[i].waiting_time;
    avg_tat += user_queue[i].turnaround_time;
    avg_resp += user_queue[i].response_time;
}
int total = sys_count + user_count;
printf("\nAverage Waiting Time: %.2f", avg_wait / total);
printf("\nAverage Turn Around Time: %.2f", avg_tat / total);
printf("\nAverage Response Time: %.2f", avg_resp / total);
printf("\nThroughput: %.2f\n", (float)total / avg_tat * total);
return 0;
}

```

RESULT:

Enter number of processes: 4
Enter Burst Time, Arrival Time and Queue of each process:

P1: 2 0 1

P2: 1 0 2

P3: 5 0 1

P4: 3 0 2

Process	Waiting Time	Turn Around Time	Response Time
P1	0	2	0
P3	2	7	2
P2	7	8	7
P4	8	11	8

Average Waiting Time: 4.25

Average Turn Around Time: 7.00

Average Response Time: 4.25

Throughput: 0.57

Process returned 0 (0x0) execution time : 15.237 s
Press any key to continue.

LAB-3:

i) Multilevel Feedback Queue Scheduling

```
#include <stdio.h>
struct Process {
    int id, bt, at, queue;
    int wt, tat, rt;
};

void roundRobin(struct Process p[], int n, int quantum) {
    int nbt = 0, time = 0;
    for (int i = 0; i < n; i++) {
        if (p[i].at <= time) {
            p[i].wt = time - p[i].at;
            p[i].tat = p[i].wt + p[i].bt;
            p[i].rt = p[i].tat - p[i].bt;
            time += p[i].bt;
        }
    }
    int nq = 0;
    int nbt;
    printf("Enter no. of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        processes[i].at = 0;
        processes[i].bt = 0;
        processes[i].queue = 0;
        processes[i].wt = 0;
        processes[i].tat = 0;
        processes[i].rt = 0;
    }
    while (nbt < n) {
        for (int i = 0; i < n; i++) {
            if (processes[i].queue == 0) {
                time += quantum;
                nbt++;
                processes[i].queue = 1;
                processes[i].wt = time - p[i].at;
                processes[i].tat = time + p[i].bt;
                processes[i].rt = processes[i].tat - processes[i].bt;
                processes[i].wt = processes[i].tat - processes[i].bt;
                processes[i].queue = 0;
                nq++;
            }
        }
        time += quantum;
        nbt++;
    }
}
```

KRUPA

```
void fcf (struct Process p[], int n, int start_time) {
    int time, start_time;
    for (int i = 0; i < n; i++) {
        if (time < p[i].at) {
            time = p[i].at;
            p[i].wt = time - p[i].at;
            p[i].tat = p[i].wt + p[i].bt;
            p[i].rt = p[i].tat - p[i].bt;
            time += p[i].bt;
        }
    }
}

int main() {
    int n;
    printf("Enter no. of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        processes[i].at = 0;
        processes[i].bt = 0;
        processes[i].queue = 0;
        processes[i].wt = 0;
        processes[i].tat = 0;
        processes[i].rt = 0;
    }
    float avg_wt = 0, avg_tat = 0, avg_rt = 0;
    for (int i = 0; i < n; i++) {
        avg_wt += processes[i].wt;
        avg_tat += processes[i].tat;
        avg_rt += processes[i].rt;
    }
    float avg_wt = avg_wt / n;
    float avg_tat = avg_tat / n;
    float avg_rt = avg_rt / n;
    printf("Avg WT: %f\n", avg_wt);
    printf("Avg TAT: %f\n", avg_tat);
    printf("Avg RT: %f\n", avg_rt);
    printf("Throughput: %f\n", n / avg_tat);
}
```

Output:

Enter no. of processes: 4
Enter Burst Time, Arrival Time and Queue of each process:
P1: 2 0 1
P2: 1 0 2
P3: 5 0 1
P4: 3 0 2

Process	WT	TAT	RT
P1	0	2	0
P2	2	7	2
P3	7	8	7
P4	8	11	8

Avg WT: 4.25
Avg TAT: 7.00
Avg RT: 4.25
Throughput: 0.57

Program -4

Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic
b) Earliest-deadline First

=> Rate Monotonic

```
#include <stdio.h>
#define MAX_PROCESSES 10
typedef struct {
    int id;
    int burst_time;
    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] = processes[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 processes[], 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].burst_time / 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", lcm_period);
printf("Rate Monotone Scheduling:\n");
printf("PID Burst Period\n");
for (int i = 0; i < n; i++) {
    printf("%d %d %d\n", processes[i].id, processes[i].burst_time, 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("\nSystem 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].burst_time;
        }
        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++;
}
int main() {
    int n;
    Process processes[MAX_PROCESSES];
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        scanf("%d", &processes[i].burst_time);
        processes[i].remaining_time = processes[i].burst_time;
    }
    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &processes[i].period);
    }
}

```

```

    }
sort_by_period(processes, n);
rate_monotonic_scheduling(processes, n);

return 0;
}

```

RESULT:

```

Enter the number of processes: 3
Enter the CPU burst times:
3 6 8
Enter the time periods:
3 4 5
LCM=60

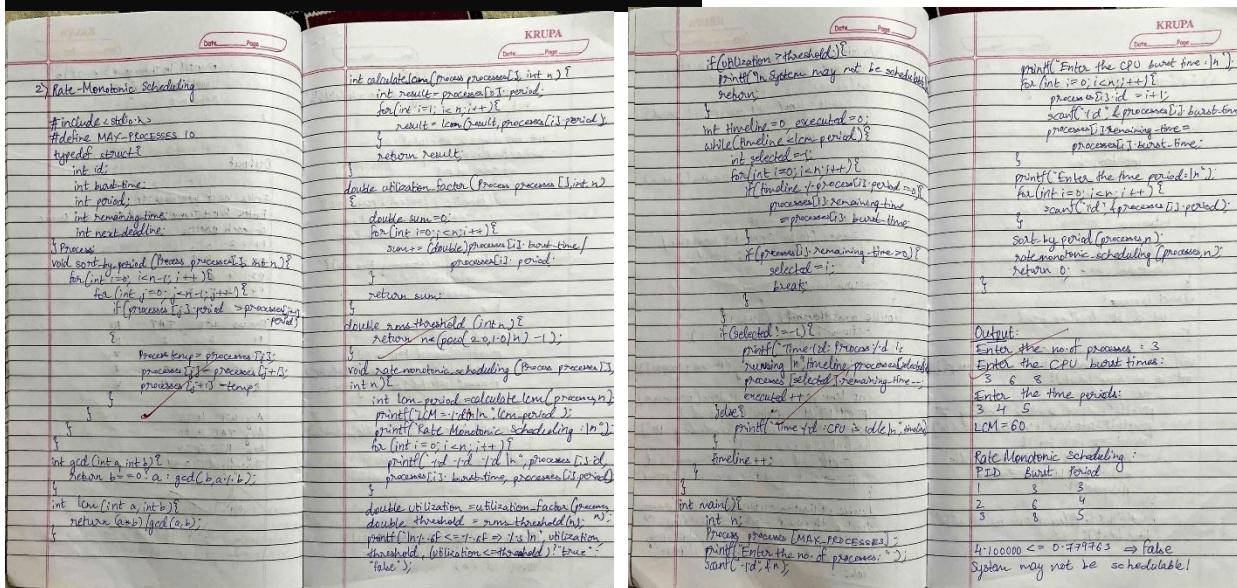
Rate Monotone Scheduling:
PID Burst Period
1 3 3
2 6 4
3 8 5

4.100000 <= 0.779763 => false

System may not be schedulable!

Process returned 0 (0x0) execution time : 34.908 s
Press any key to continue.

```



=> 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%d\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;
}

```

RESULT:

```

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 : 9.656 s
Press any key to continue.

```

3) Earliest-Deadline scheduling

```

#include <stdio.h>
int gcd(int a, int b) {
    while (a != 0) {
        if (a > b) {
            int temp = a;
            a = b;
            b = temp;
        }
        a = a % b;
    }
    return a;
}

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

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

void earliest_deadline_first(Process p[], int n, int time, int& t) {
    int i;
    for (i = 0; i < n; i++) {
        if (p[i].deadline <= time) {
            p[i].id = t;
            t++;
        }
    }
}

int main() {
    int n;
    printf("Enter the no. of processes : ");
    scanf("%d", &n);
    struct Process p[n];
    printf("Enter the CPU burst times : ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &p[i].bt);
    }
    printf("Enter the deadlines : ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &p[i].deadline);
    }
    printf("Enter the time periods : ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &p[i].period);
    }

    printf("In scheduling occurs for %d ms\n", time);
    time = lcm(p[0].period, p[1].period);
    earliest_deadline_first(p, n, time);
    printf("Hyperperiod : %d ms\n", time);
}

```

KRUPA

```

printf("Enter the time periods : \n");
for (int i = 0; i < n; i++) {
    scanf("%d", &p[i].period);
}
int hyperperiod = p[0].period;
for (int i = 0; i < n; i++) {
    hyperperiod = lcm(hyperperiod, p[i].period);
}
printf("System will execute for hyperperiod (LCM of periods): %d ms\n", hyperperiod);
earliest_deadline_first(p, n, hyperperiod);
return 0;
}

Output:
Enter the no. 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:

```

Program 5

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

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 3;
int x = 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("Producer produces item %d\n", x);
    mutex = signal(mutex);
}
void consumer() {
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf("Consumer consumes item %d\n", x);
    x--;
    mutex = signal(mutex);
}
int main() {
    int choice;
    while (1) {
        printf("\n1. Producer\n2. Consumer\n3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            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);

default:
    printf("Invalid choice!\n");
}

}

return 0;
}

```

RESULT:

```

1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer produces item 1

1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer produces item 2

1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer produces item 3

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

1. Producer
2. Consumer
3. Exit
Enter your choice: 2
Consumer consumes item 3

1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer produces item 3

1. Producer
2. Consumer
3. Exit
Enter your choice: 2
Consumer consumes item 3

1. Producer
2. Consumer
3. Exit
Enter your choice: 2
Consumer consumes item 2

1. Producer
2. Consumer
3. Exit
Enter your choice: 2
Consumer consumes item 1

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

```

KRUPA

10/21/25

LAB-4

1) Producer - Consumer

```

#include<stdio.h>
#include<stdlib.h>
int mutex = 1;
int full = 0;
int empty = 3;
int x = 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("Producer produces item %d\n", x);
    mutex = signal(mutex);
}
void consumer() {
    mutex = wait(mutex);
    full = wait(full);
    empty = signal(empty);
    printf("Consumer consumes item %d\n", x);
    x--;
    mutex = signal(mutex);
}

```

int main() {
 int choice;
 while(1) {
 printf("1. Producer\n2. Consumer\n3. Exit\n");
 printf("Enter your choice: ");
 scanf("%d", &choice);
 switch(choice) {
 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);
 default:
 printf("Invalid choice!\n");
 }
 }
 return 0;
}

2)

Output:

```

1. Producer
2. Consumer
3. Exit
Enter your choice: 1
Producer produces item 1

Enter your choice: 1
Producer produces item 2

Enter your choice: 1
Producer produces item 3

Enter your choice: 2
Buffer is full

Enter your choice: 2
Consumer consumes item 3

Enter your choice: 2
Consumer consumes item 2

Enter your choice: 1
Consumer consumes item 1

Enter your choice: 2
Buffer is empty!

```

Program 6

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

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int totalPhilosophers;
int hungry[MAX];
int areNeighbors(int a, int b) {
    return (abs(a - b) == 1 || abs(a - b) == totalPhilosophers - 1);
}
void option1(int count) {
    printf("\nAllow one philosopher to eat at any time\n");

    for (int i = 0; i < count; i++) {
        printf("P %d is granted to eat\n", hungry[i]);
        for (int j = 0; j < count; j++) {
            if (j != i) {
                printf("P %d is waiting\n", hungry[j]);
            }
        }
    }
}
void option2(int count) {
    printf("\nAllow two philosophers to eat at same time\n");
    int combination = 1;
    for (int i = 0; i < count; i++) {
        for (int j = i + 1; j < count; j++) {
            if (!areNeighbors(hungry[i], hungry[j])) {
                printf("combination %d\n", combination++);
                printf("P %d and P %d are granted to eat\n", hungry[i], hungry[j]);
                for (int k = 0; k < count; k++) {
                    if (k != i && k != j) {
                        printf("P %d is waiting\n", hungry[k]);
                    }
                }
                printf("\n");
            }
        }
    }
    if (combination == 1) {
        printf("No combinations found where two non-neighbor philosophers can eat.\n");
    }
}
int main() {
    int hungryCount;
    printf("DINING PHILOSOPHER PROBLEM\n");
    printf("Enter the total no. of philosophers: ");
    scanf("%d", &totalPhilosophers);
    printf("How many are hungry: ");
    scanf("%d", &hungryCount);
```

```

for (int i = 0; i < hungryCount; i++) {
    printf("Enter philosopher %d position: ", i + 1);
    scanf("%d", &hungry[i]);
}
int choice;
do {
    printf("\n1. One can eat at a time  2. Two can eat at a time  3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            option1(hungryCount);
            break;
        case 2:
            option2(hungryCount);
            break;
        case 3:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
    }
} while (choice != 3);

return 0;
}

```

RESULT:

```

DINING PHILOSOPHER PROBLEM
Enter the total no. of philosophers: 5
How many are hungry: 3
Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5

1. One can eat at a time   2. Two can eat at a time   3. Exit
Enter your choice: 1

Allow one philosopher to eat at any time
P 2 is granted to eat
P 4 is waiting
P 5 is waiting
P 4 is granted to eat
P 2 is waiting
P 5 is waiting
P 5 is granted to eat
P 2 is waiting
P 4 is waiting

1. One can eat at a time   2. Two can eat at a time   3. Exit
Enter your choice: 2

Allow two philosophers to eat at same time
combination 1
P 2 and P 4 are granted to eat
P 5 is waiting

combination 2
P 2 and P 5 are granted to eat
P 4 is waiting

1. One can eat at a time   2. Two can eat at a time   3. Exit
Enter your choice: 3
Exiting...

Process returned 0 (0x0)  execution time : 30.628 s
Press any key to continue.

```

KRUPA

2) Dining-Philosopher

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
int totalPhilosophers;
int hungry[MAX];
int areNeighbors(int a, int b) {
    return (abs(a-b)==1 || abs(a-b)==totalPhilosophers-1);
}
void option1(int count) {
    printf("In Allow one philosopher to eat at any time\n");
    for (int i=0; i<count; i++) {
        printf("P-%d is granted to eat\n", hungry[i]);
        for (int j=0; j<count; j++) {
            if (j!=i) {
                printf("P-%d is waiting\n", hungry[j]);
            }
        }
    }
}
void option2(int count) {
    printf("In Allow two philosophers to eat at same time\n");
    int combination = 1;
    for (int i=0; i<count; i++) {
        for (int j=i+1; j<count; j++) {
            combination++;
        }
    }
}
```

KRUPA

```
if (!areNeighbors(hungry[i], hungry[j])) {
    printf("combination-%d\n", combination);
    printf("P-%d and P-%d are granted to eat\n", hungry[i], hungry[j]);
    for (int k=0; k<count; k++) {
        if (k!=i && k!=j) {
            printf("P-%d is waiting\n", hungry[k]);
        }
    }
}
printf("\n");
if (combination == 1) {
    printf("No combination found where two non-neighbor philosophers can eat.\n");
}
int main() {
    int hungryCount;
    printf("Enter the total no. of philosophers:");
    scanf("%d", &totalPhilosophers);
    printf("How many are hungry: ");
    scanf("%d", &hungryCount);
    for (int i=0; i<hungryCount; i++) {
        printf("Enter philosopher %d position: ", i+1);
        scanf("%d", &hungry[i]);
    }
    int choice;
```

Output:

Dining Philosopher Problem
 Enter the total no. of philosophers: 5
 How many are hungry: 3
 Enter philosopher 1 position: 2
 Enter philosopher 2 position: 4
 Enter philosopher 3 position: 5

KRUPA

```
1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 1
Allow one philosopher to eat at any time
P 2 is granted to eat
P 4 is waiting
P 5 is waiting
P 4 is granted to eat
P 2 is waiting
P 5 is granted to eat
P 2 is waiting
P 4 is waiting
1. One can eat at a time
2. Two can eat at a time
3. Exit
Enter your choice: 2
Allow two philosophers to eat at same time
combination 1
P 2 and P 4 are granted to eat
P 5 is waiting
combination 2
P 2 and P 5 are granted to eat
P 4 is waiting
```

Program 7

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

```
#include <stdio.h>
#include <stdbool.h>
int main() {
    int n, m;
    printf("Enter number of processes and resources:\n");
    scanf("%d %d", &n, &m);
    int alloc[n][m], max[n][m], avail[m];
    printf("Enter allocation matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            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 i = 0; i < m; i++)
        scanf("%d", &avail[i]);
    int need[n][m];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    bool finish[n];
    for (int i = 0; i < n; i++)
        finish[i] = false;
    int safeSeq[n];
    int work[m];
    for (int i = 0; i < m; i++)
        work[i] = avail[i];
    int count = 0;
    while (count < n) {
        bool found = false;
        for (int p = 0; p < n; p++) {
            if (!finish[p]) {
                int j;
                for (j = 0; j < m; j++)
                    if (need[p][j] > work[j])
                        break;
                if (j == m) {
                    for (int k = 0; k < m; k++)
                        work[k] += alloc[p][k];

                    safeSeq[count++] = p;
                    finish[p] = true;
                    found = true;
                }
            }
        }
    }
}
```

```

if (!found) {
    printf("System is not in a safe state.\n");
    return 0;
}
printf("System is in safe state.\nSafe sequence is: ");
for (int i = 0; i < n; i++)
    printf("P%d%s", safeSeq[i], (i == n - 1) ? "\n" : " -> ");
return 0;
}

```

RESULT:

```

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 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 state.
Safe sequence is: P1 -> P3 -> P4 -> P0 -> P2

Process returned 0 (0x0)   execution time : 77.126 s
Press any key to continue.

```

7/04/25

LAB-5.

1) Banker's Algorithm for the purpose of deadlock avoidance.

```

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

KRUPA

```

int work(m).
for(int i=0; i<m; i++)
    work[i]=avail[i];
int count = 0;
while (Count < n)
{
    bool found = false;
    for(int p=0; p<n; p++)
    {
        if (!finish[p])
        {
            int j;
            for(j=0; j<m; j++)
                if(need[p][j] > work[j])
                    break;
            if (j == m)
            {
                for(int k=0; k<n; k++)
                    work[k] += alloc[p][k];
                safeSeq[Count] = p;
                finish[p] = true;
                found = true;
            }
        }
    }
    if (!found)
        printf("System is not in a safe state.");
    return 0;
}
printf("System is in safe state.\nSafe sequence is: ");
for(int i = 0; i < n; i++)
    printf("P%d%s", safeSeq[i], (i == n - 1) ? "\n" : " -> ");
return 0;
}

```

KRUPA

```

Enter the 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 state.
Safe sequence is: P1 -> P3 -> P4 -> P0 -> P2

```

Program 8

Write a C program to simulate deadlock detection

```
#include <stdio.h>
#include <stdbool.h>
int main() {
    int n, m;
    printf("Enter number of processes and number of resources:\n");
    scanf("%d %d", &n, &m);
    int alloc[n][m], request[n][m], avail[m];
    printf("Enter Allocation Matrix (%d x %d):\n", n, m);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);
    printf("Enter Request Matrix (%d x %d):\n", n, m);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &request[i][j]);

    printf("Enter Available Resources (%d values):\n", m);
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);
    int work[m];
    for (int i = 0; i < m; i++)
        work[i] = avail[i];
    bool finish[n];
    for (int i = 0; i < n; i++) {
        bool hasAllocation = false;
        for (int j = 0; j < m; j++) {
            if (alloc[i][j] != 0) {
                hasAllocation = true;
                break;
            }
        }
        finish[i] = hasAllocation ? false : true;
    }
    while (true) {
        bool progress = false;

        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                bool canGrant = true;

                for (int j = 0; j < m; j++) {
                    if (request[i][j] > work[j]) {
                        canGrant = false;
                        break;
                    }
                }
                if (canGrant) {
```

```

        for (int j = 0; j < m; j++)
            work[j] += alloc[i][j];
        finish[i] = true;
        progress = true;
    }
}
}

if (!progress)
    break;
}

printf("\nDeadlock Detection Result:\n");
bool deadlock = false;

for (int i = 0; i < n; i++) {
    if (!finish[i]) {
        printf("Process P%d is DEADLOCKED\n", i);
        deadlock = true;
    } else {
        printf("Process P%d is NOT deadlocked\n", i);
    }
}
if (!deadlock)
    printf("\nNo deadlock detected in the system.\n");

return 0;
}

```

RESULT:

```

Enter number of processes and number of resources:
5 3
Enter Allocation Matrix (5 x 3):
0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
Enter Request Matrix (5 x 3):
0 0 0
2 0 2
0 0 1
1 0 0
0 0 2
Enter Available Resources (3 values):
0 0 0

Deadlock Detection Result:
Process P0 is NOT deadlocked
Process P1 is DEADLOCKED
Process P2 is DEADLOCKED
Process P3 is DEADLOCKED
Process P4 is DEADLOCKED

Process returned 0 (0x0)   execution time : 44.041 s
Press any key to continue.

```

2) Deadlock Detection

```

#include <stdio.h>
#include <stdbool.h>
int main()
{
    int n,m;
    printf("Enter no. of processes and
resources:\n");
    scanf("%d %d", &n, &m);

```

```

KRUPA
Date _____ Page _____
int alloc[n][m], request[n][m], avail[m];
printf("Enter Allocation Matrix (n x m)\n");
for(int i=0; i<n; i++)
    for(int j=0; j<m; j++)
        scanf("%d", &alloc[i][j]);
printf("Enter Request Matrix (n x m)\n");
for(int i=0; i<n; i++)
    for(int j=0; j<m; j++)
        scanf("%d", &request[i][j]);
printf("Enter Available Resources (m values)\n");
for(int i=0; i<m; i++)
    scanf("%d", &avail[i]);
int work[m];
for(int i=0; i<n; i++){
    work[i] = avail[i];
}
bool finish[n];
for(int i=0; i<n; i++) {
    bool hasAllocation = false;
    for(int j=0; j<m; j++) {
        if(alloc[i][j] != 0) {
            hasAllocation = true;
            break;
        }
    }
    finish[i] = hasAllocation ? false : true;
}
while(true) {
    bool progress = false;
    for(int i=0; i<n; i++) {
        if(finish[i])
            continue;
        for(int j=0; j<m; j++) {
            if(request[i][j] > work[j]) {
                continue;
            }
            work[j] -= request[i][j];
            if(work[j] == avail[j]) {
                finish[i] = true;
                progress = true;
            }
        }
    }
    if(!progress)
        break;
}
printf("Deadlock Detection Result:\n");
for(int i=0; i<n; i++) {
    if(finish[i])
        printf("Process P%d is Deadlocked\n", i+1);
    else
        printf("Process P%d is NOT Deadlocked\n", i+1);
}

```

```

Date _____ Page _____
if(!deadlock)
    printf("No deadlock detected
    in the system\n")
return 0;

Output:
Enter no of processes and resources:
5 3
Enter Allocation matrix (5 x 3):
0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
Enter Request Matrix (5 x 3):
0 0 0
2 0 2
0 0 1
1 0 0
0 0 2
Enter Available Resources (3 values):
0 0 0
Deadlock Detection Result:
Process P0 is NOT Deadlocked
Process P1 is Deadlocked
Process P2 is Deadlocked
Process P3 is Deadlocked
Process P4 is Deadlocked.

```

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>
struct Block {
    int size;
    int allocated;
};

struct File {
    int size;
    int block_no;
};

void resetBlocks(struct Block blocks[], 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\tMemory Management Scheme – First Fit\n");
    printf("File_no:\tFile_size\tBlock_no:\tBlock_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[j].size >= files[i].size) {
                files[i].block_no = j + 1;
                blocks[j].allocated = 1;
                printf("%d\t%d\t%d\t%d\n", i + 1, files[i].size, j + 1, blocks[j].size);
                break;
            }
        }
        if (files[i].block_no == -1) {
            printf("%d\t%d\t-\t-\n", i + 1, files[i].size);
        }
    }
}

void bestFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\n\tMemory Management Scheme – Best Fit\n");
    printf("File_no:\tFile_size\tBlock_no:\tBlock_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%d\t%d\t%d\n", i + 1, files[i].size, bestIdx + 1, blocks[bestIdx].size);
    } else {
        printf("%d\t%d\t%d\t-\n", i + 1, files[i].size);
    }
}
}

void worstFit(struct Block blocks[], int n_blocks, struct File files[], int n_files) {
    printf("\nMemory Management Scheme – Worst Fit\n");
    printf("File_no:\tFile_size\tBlock_no:\tBlock_size:\n");

    for (int i = 0; i < n_files; 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%d\t%d\t%d\t%d\n", i + 1, files[i].size, worstIdx + 1,
blocks[worstIdx].size);
        } else {
            printf("%d\t%d\t%d\t-\n", i + 1, files[i].size);
        }
    }
}

int main() {
    int n_blocks, n_files, choice;

    printf("Memory Management Scheme\n");

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

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

```

```

struct Block blocks[n_blocks];
struct File files[n_files];

printf("\nEnter the size of the blocks:\n");
for (int i = 0; i < n_blocks; i++) {
    printf("Block %d: ", i + 1);
    scanf("%d", &blocks[i].size);
    blocks[i].allocated = 0;
}

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

do {
    printf("\n1. First Fit\n2. Best Fit\n3. Worst Fit\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    resetBlocks(blocks, n_blocks); // Reset block allocation before each strategy

    switch (choice) {
        case 1:
            firstFit(blocks, n_blocks, files, n_files);
            break;
        case 2:
            bestFit(blocks, n_blocks, files, n_files);
            break;
        case 3:
            worstFit(blocks, n_blocks, files, n_files);
            break;
        case 4:
            printf("\nExiting...\n");
            break;
        default:
            printf("Invalid choice.\n");
    }
} while (choice != 4);

return 0;
}

```

RESULT:

```

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 @ 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 @ 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 @ 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: -

```

KRUPA

15/05/25 LAB-6

i) Memory Allocation.

```

#include<stdio.h>
struct Block {
    int size;
    int allocated;
};

struct File {
    int size;
    int block_no;
};

void reset_blocks(struct Block blocks[], int n) {
    for (int i=0; i<n; i++) {
        blocks[i].allocated = 0;
    }
}

void firstfit (struct Block blocks [5], int nblocks, struct File files [5], int nfiles) {
    printf("IntMemory Management scheme\n");
    printf("-FirstFit\n");
    printf("File-no: 1File_size[1]Block_no[1]\n");
    printf("Block_size[1]\n");
    for (int i=0; i<nfiles; i++) {
        files[i].block_no = -1;
        for (int j=0; j<nblocks; j++) {
            if (!blocks[j].allocated && blocks[j].size >= files[i].size) {
                files[i].block_no = j + 1;
                blocks[j].allocated = 1;
                printf("1d1b[1]d1f[1]d1t[1]\n");
                printf("1d1n[1], file[%d].size[%d],\n", i+1,
                    blocks[j].size);
            }
        }
    }
}

```

KRUPA

```

break;
}
if (file[i].blockno == -1) {
    printf("File %d Blockno = -1\n", i+1, file[i].size);
}

void bestFit(struct Block blocks[], int n_blocks,
            struct File files[], int n_files) {
    printf("Best Fit\n");
    printf("File no\tFile size\tBlock no\tBlock size\n");
    for (int i = 0; i < n; i++) {
        int best_idx = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (blocks[j].allocated == 0) {
                if (blocks[j].size >= files[i].size) {
                    if (best_idx == -1 || blocks[j].size < blocks[best_idx].size)
                        best_idx = j;
                }
            }
        }
        if (best_idx != -1) {
            blocks[best_idx].allocated = 1;
            files[i].block_no = best_idx + 1;
            printf("File %d Block no = %d\n", i+1, best_idx+1);
            files[i].size = blocks[best_idx].size;
            blocks[best_idx].size = 0;
        }
    }
}

void worstFit(struct Block blocks[], int n_blocks,
             struct File files[], int n_files) {
    printf("Worst Fit\n");
    printf("File no\tFile size\tBlock no\tBlock size\n");
    for (int i = 0; i < n; i++) {
        int worst_idx = -1;
        for (int j = 0; j < n_blocks; j++) {
            if (blocks[j].allocated == 0) {
                if (blocks[j].size == files[i].size) {
                    if (worst_idx == -1 || blocks[j].size > blocks[worst_idx].size)
                        worst_idx = j;
                }
            }
        }
        if (worst_idx != -1) {
            blocks[worst_idx].allocated = 1;
            files[i].block_no = worst_idx + 1;
            printf("File %d Block no = %d\n", i+1, worst_idx+1);
            files[i].size = blocks[worst_idx].size;
            blocks[worst_idx].size = 0;
        }
    }
}

void roundRobin(struct Block blocks[], int n_blocks,
                struct File files[], int n_files, int quantum) {
    printf("Round Robin\n");
    printf("File no\tFile size\tBlock no\tBlock size\n");
    for (int i = 0; i < n; i++) {
        int current_idx = i;
        int current_size = files[i].size;
        while (current_size > 0) {
            if (blocks[current_idx].size >= current_size) {
                blocks[current_idx].allocated = 1;
                files[i].block_no = current_idx + 1;
                printf("File %d Block no = %d\n", i+1, current_idx+1);
                files[i].size = current_size;
                blocks[current_idx].size = 0;
                break;
            } else {
                current_idx = (current_idx + 1) % n_blocks;
                current_size -= blocks[current_idx].size;
            }
        }
    }
}

```

KRUPA

```

int main() {
    int n_blocks, n_files;
    printf("Memory Management Scheme\n");
    printf("Enter no of blocks & no of files : ");
    scanf("%d %d", &n_blocks, &n_files);
    struct Block blocks[n_blocks];
    struct File files [n_files];
    printf("Enter size of blocks : ");
    for (int i = 0; i < n_blocks; i++) {
        printf("Block %d : ", i+1);
        scanf("%d", &blocks[i].size);
        blocks[i].allocated = 0;
    }
    printf("Enter the size of the file : ");
    for (int i = 0; i < n_files; i++) {
        printf("File %d : ", i+1);
        scanf("%d", &files[i].size);
    }
    do {
        printf("1. First Fit\n");
        printf("2. Best Fit\n");
        printf("3. Worst Fit\n");
        printf("4. Exit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                firstFit(blocks, n_blocks, files, n_files);
                break;
            case 2:
                bestFit(blocks, n_blocks, files, n_files);
                break;
            case 3:
                worstFit(blocks, n_blocks, files, n_files);
                break;
            case 4:
                exit(0);
        }
    } while (choice != 4);
}

Output:
Memory Management scheme
Enter the no of blocks:5
Enter the no of files:4
Enter size of blocks:
Block1: 100
Block2: 500
Block3: 200
Block4: 300
Block5: 600
Enter size of 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 in First Fit			
File no:	File size	Block no:	Block size:
1	212	2	500
2	417	5	600
3	112	3	200
4	420	-	-

Memory Management Scheme in 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

Memory Management Scheme in Worst Fit			
File no:	File size	Block no:	Block size:
1	212	5	600
2	417	2	500
3	112	4	300
4	420	-	-

Program 10

Write a C program to simulate page replacement algorithms

- a) FIFO
- b) LRU
- c) Optimal

=>FIFO

```
#include <stdio.h>

int main() {
    int frames, pages[50], n, frame[10], i, j, k, avail, count = 0;

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

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

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

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

    printf("\nPage\tFrames\tPage 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("\tYes\n");
}
```

```

} else {
    printf("%d\t", pages[i]);
    for(k = 0; k < frames; k++) {
        if(frame[k] != -1)
            printf("%d ", frame[k]);
        else
            printf("- ");
    }
    printf("\tNo\n");
}
}

printf("\nTotal Page Faults = %d\n", count);
return 0;
}

```

RESULT:

```

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
4      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 : 28.979 s
Press any key to continue.

```

=>LRU:

```

#include <stdio.h>

int main() {
    int n, frames, i, j, k, faults = 0;
    printf("Enter number 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(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 min_time = time[0], min_pos = 0;
        for(k = 1; k < frames; k++) {
            if(time[k] < min_time) {
                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);
return 0;
}

```

RESULT:

```
Enter number of pages: 20
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of frames: 4
Frames after accessing 7: 7 - - -
Frames after accessing 0: 7 0 - -
Frames after accessing 1: 7 0 1 -
Frames after accessing 2: 7 0 1 2
Frames after accessing 0: 7 0 1 2
Frames after accessing 3: 3 0 1 2
Frames after accessing 0: 3 0 1 2
Frames after accessing 4: 3 0 4 2
Frames after accessing 2: 3 0 4 2
Frames after accessing 3: 3 0 4 2
Frames after accessing 0: 3 0 4 2
Frames after accessing 3: 3 0 4 2
Frames after accessing 2: 3 0 4 2
Frames after accessing 1: 3 0 1 2
Frames after accessing 2: 3 0 1 2
Frames after accessing 0: 3 0 1 2
Frames after accessing 1: 3 0 1 2
Frames after accessing 7: 7 0 1 2
Frames after accessing 0: 7 0 1 2
Frames after accessing 1: 7 0 1 2
Total page faults: 8
Total page Hits: 12

Process returned 0 (0x0)   execution time : 54.602 s
Press any key to continue.
```

=>OPTIMAL

```
#include <stdio.h>
int main() {
    int n, frames, i, j, k, faults = 0;
    printf("Enter number 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];
    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)
                frame_arr[pos] = pages[i];
        }
    }
}
```

```

        }
    }
    if(pos == -1) {
        int farthest = i, replace_index = 0;
        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("%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);
return 0;
}

```

RESULT:

```

Enter number of pages: 20
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of frames: 4
7: 7 0 - -
0: 7 0 1 -
1: 7 0 1 2
2: 7 0 1 2
0: 7 0 1 2
3: 3 0 1 2
0: 3 0 1 2
4: 3 0 4 2
2: 3 0 4 2
3: 3 0 4 2
0: 3 0 4 2
3: 3 0 4 2
2: 3 0 4 2
1: 1 0 4 2
2: 1 0 4 2
0: 1 0 4 2
1: 1 0 4 2
7: 1 0 7 2
0: 1 0 7 2
1: 1 0 7 2
Total page faults: 8
Total page Hits: 12
Process returned 0 (0x0) execution time : 29.373 s
Press any key to continue.

```

1 AB-7

Page Replacement - FIFO

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

Page Frames Page Faults

#	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------