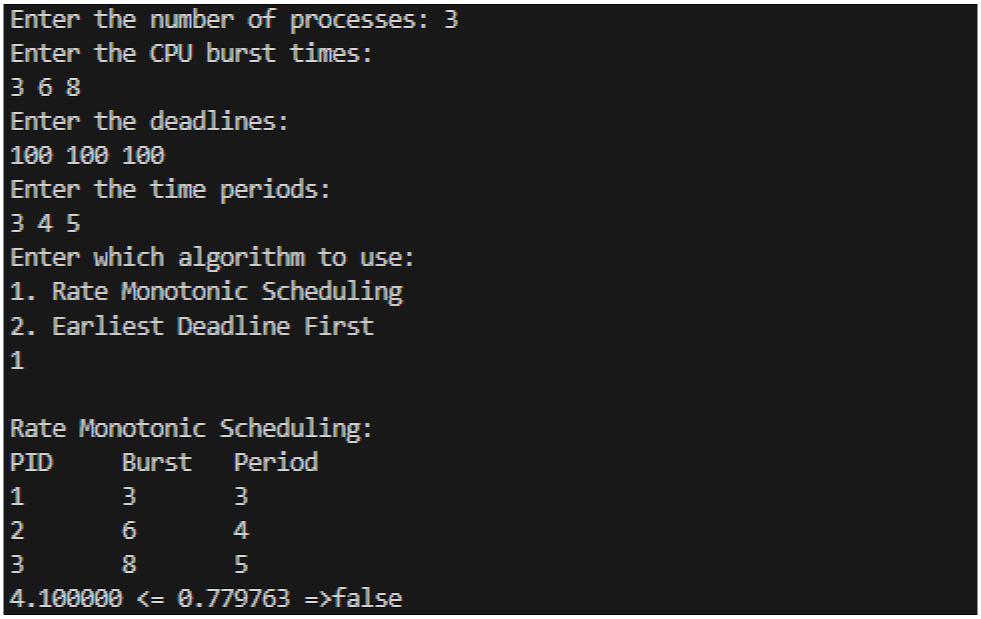
Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic
b) Earliest-deadline First

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

#define MAX 10

struct Task {
    int id, burst, period, deadline;
    int remaining, next_deadline;
};

int lcm(int a, int b) {
    int max = (a > b) ? a : b;
    while (1) {
        if (max % a == 0 && max % b == 0)
            return max;
        ++max;
    }
}

int lcm_multiple(int arr[], int n) {
    int res = arr[0];
    for (int i = 1; i < n; i++)
        res = lcm(res, arr[i]);
    return res;
}

void rate_monotonic(struct Task tasks[], int n) {
    int periods[MAX];
    for (int i = 0; i < n; i++)
        periods[i] = tasks[i].period;

    int l = lcm_multiple(periods, n);
    printf("\nRate Monotonic Scheduling:\n");
    printf("PID\tBurst\tPeriod\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\n", tasks[i].id, tasks[i].burst, tasks[i].period);

    float utilization = 0;
    for (int i = 0; i < n; i++)
        utilization += (float)tasks[i].burst / tasks[i].period;

    float bound = n * (pow(2.0, 1.0 / n) - 1);
    printf("%.6f <= %.6f =>%s\n", utilization, bound, (utilization <= bound) ? "true" :
"false");

    for (int t = 0; t < l; t++) {
        for (int i = 0; i < n; i++) {
            if (t % tasks[i].period == 0)
                tasks[i].remaining = tasks[i].burst;
        }

        int current = -1;
        for (int i = 0; i < n; i++) {
            if (tasks[i].remaining > 0) {
                if (current == -1 || tasks[i].period < tasks[current].period)
                    current = i;
            }
        }

        if (current != -1)
            tasks[current].remaining--;
    }
}

void earliest_deadline_first(struct Task tasks[], int n) {
    int periods[MAX];
    for (int i = 0; i < n; i++)
        periods[i] = tasks[i].period;

    int l = lcm_multiple(periods, n);
    printf("\nEarliest Deadline Scheduling:\n");
    printf("PID\tBurst\tDeadline\n");
    for (int i = 0; i < n; i++)
        printf("%d\t%d\t%d\n", tasks[i].id, tasks[i].burst, tasks[i].deadline);
    printf("Scheduling occurs for %d ms\n\n", l);

    for (int t = 0; t < l; t++) {
        for (int i = 0; i < n; i++) {
            if (t % tasks[i].period == 0) {
                tasks[i].remaining = tasks[i].burst;
                tasks[i].next_deadline = t + tasks[i].deadline;
            }
        }

        int current = -1;
        for (int i = 0; i < n; i++) {
            if (tasks[i].remaining > 0) {
                if (current == -1 || tasks[i].next_deadline < tasks[current].next_deadline)
                    current = i;
            }
        }

        if (current != -1) {
            printf("%dms : Task %d is running.\n", t, tasks[current].id);
            tasks[current].remaining--;
        } else {
            printf("%dms : CPU is idle.\n", t);
        }
    }
}

int main() {
    int n;
    struct Task tasks[MAX], rms_tasks[MAX], edf_tasks[MAX];

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

    printf("Enter the CPU burst times:\n");
    for (int i = 0; i < n; i++)
```

```c
        scanf("%d", &tasks[i].burst);

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

    printf("Enter the time periods:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &tasks[i].period);
        tasks[i].id = i + 1;
    }

    for (int i = 0; i < n; i++) {
        rms_tasks[i] = tasks[i];
        rms_tasks[i].remaining = 0;
        edf_tasks[i] = tasks[i];
        edf_tasks[i].remaining = 0;
        edf_tasks[i].next_deadline = 0;
    }
    int q;
    printf("Enter which algorithm to use: \n");
    printf("1. Rate Monotonic Scheduling\n");
    printf("2. Earliest Deadline First\n");
    scanf("%d",&q);
    if(q==1){
        rate_monotonic(rms_tasks, n);
    }
    else{
        earliest_deadline_first(edf_tasks, n);
    }



    return 0;
}
```

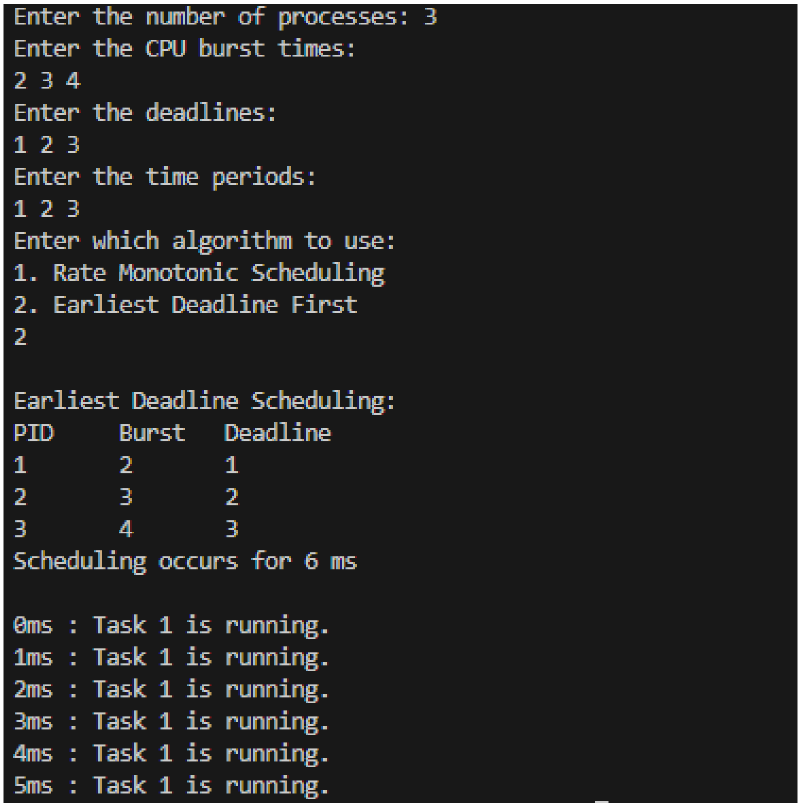
OUTPUT

a) Rate- Monotonic



```
Enter the number of processes: 3
Enter the CPU burst times:
3 6 8
Enter the deadlines:
100 100 100
Enter the time periods:
3 4 5
Enter which algorithm to use:
1. Rate Monotonic Scheduling
2. Earliest Deadline First
1

Rate Monotonic Scheduling:
PID     Burst   Period
1       3       3
2       6       4
3       8       5
4.100000 <= 0.779763 =>false
```

b) Earliest-deadline First



```
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
Enter which algorithm to use:
1. Rate Monotonic Scheduling
2. Earliest Deadline First
2

Earliest Deadline Scheduling:
PID     Burst   Deadline
1       2       1
2       3       2
3       4       3
Scheduling occurs for 6 ms

0ms : Task 1 is running.
1ms : Task 1 is running.
2ms : Task 1 is running.
3ms : Task 1 is running.
4ms : Task 1 is running.
5ms : Task 1 is running.
```