

```

#include <limits.h>
#include <stdio.h>

#define MAX 10

typedef struct {
    int id;
    int arrival_time;
    int burst_time;
    int priority;
    int remaining_time;
    int turnaround_time;
    int waiting_time;
} Process;

void round_robin(Process processes[], int n, int quantum) {
    int time = 0, i, flag;
    int remaining_processes = n;

    while (remaining_processes > 0) {
        flag = 0;
        for (i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time &&
                processes[i].remaining_time > 0) {
                flag = 1;
                if (processes[i].remaining_time <= quantum) {
                    time += processes[i].remaining_time;
                    processes[i].turnaround_time = time - processes[i].arrival_time;
                    processes[i].waiting_time =
                        processes[i].turnaround_time - processes[i].burst_time;
                    processes[i].remaining_time = 0;
                    remaining_processes--;
                } else {
                    time += quantum;
                    processes[i].remaining_time -= quantum;
                }
            }
        }
        if (flag == 0) {
            time++;
        }
    }
}

void priority_scheduling(Process processes[], int n) {
    int time = 0, i, min_priority_index;
    int completed = 0;

    while (completed < n) {
        min_priority_index = -1;
        for (i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time &&
                processes[i].remaining_time > 0) {
                if (min_priority_index == -1 ||
                    processes[i].priority < processes[min_priority_index].priority) {
                    min_priority_index = i;
                }
            }
        }
        if (min_priority_index != -1) {
            processes[min_priority_index].remaining_time--;
            time++;
            if (processes[min_priority_index].remaining_time == 0) {

```

```

        processes[min_priority_index].turnaround_time =
            time - processes[min_priority_index].arrival_time;
        processes[min_priority_index].waiting_time =
            processes[min_priority_index].turnaround_time -
            processes[min_priority_index].burst_time;
        completed++;
    }
} else {
    time++;
}
}
}

void print_results(Process processes[], int n, const char *algorithm) {
    int i;
    float total_turnaround_time = 0, total_waiting_time = 0;
    printf("%s Scheduling Results:\n", algorithm);
    printf("ID\tArrival\tBurst\tPriority\tTurnaround\tWaiting\n");
    for (i = 0; i < n; i++) {
        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;
        printf("%d\t%d\t%d\t%d\t\t%d\t\t%d\n", processes[i].id,
            processes[i].arrival_time, processes[i].burst_time,
            processes[i].priority, processes[i].turnaround_time,
            processes[i].waiting_time);
    }
    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
}

int main() {
    int n, i, quantum;
    Process processes[MAX];

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

    for (i = 0; i < n; i++) {
        printf("Enter arrival time, burst time, and priority for process %d: ",
            i + 1);
        processes[i].id = i + 1;
        scanf("%d %d %d", &processes[i].arrival_time, &processes[i].burst_time,
            &processes[i].priority);
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].turnaround_time = 0;
        processes[i].waiting_time = 0;
    }

    printf("Enter time quantum for Round Robin: ");
    scanf("%d", &quantum);

    Process rr_processes[MAX];
    Process pr_processes[MAX];

    for (i = 0; i < n; i++) {
        rr_processes[i] = processes[i];
        pr_processes[i] = processes[i];
    }

    round_robin(rr_processes, n, quantum);
    print_results(rr_processes, n, "Round Robin");

    priority_scheduling(pr_processes, n);
    print_results(pr_processes, n, "Priority");
}

```

```
    return 0;  
}
```