#include <stdio.h>  
  
struct Process {  
    int pid;  
    int priority;  
    int burst\_time;  
    int arrival\_time;  
    int start\_time;  
    int completion\_time;  
    int waiting\_time;  
    int turnaround\_time;  
    int remaining\_time;  
    int executed\_time;  
};  
  
void displayGanttChart(struct Process processes[], int n) {  
    int i, j;  
    char ganttChart[100];  
  
    for (i = 0, j = 0; i < n; ++i) {  
        if (i > 0 && processes[i].start\_time > processes[i - 1].completion\_time) {  
            while (j < processes[i].start\_time) {  
                ganttChart[j++] = '-';  
            }  
        }  
  
        while (j < processes[i].completion\_time) {  
            ganttChart[j++] = '#';  
        }  
    }  
  
    printf("Gantt Chart:\n");  
  
    for (i = 0; i < j; ++i) {  
        printf("%c", ganttChart[i]);  
    }  
  
    printf("\n");  
}  
  
void calculateWaitingTime(struct Process processes[], int n) {  
    int currentTime, i, j;  
  
    for (i = 0; i < n; ++i) {  
        currentTime = processes[i].start\_time;  
        processes[i].waiting\_time = 0;  
  
        for (j = 0; j < n; ++j) {  
            if (processes[j].arrival\_time < currentTime || (processes[j].arrival\_time == currentTime && processes[j].priority < processes[i].priority)) {  
                processes[i].waiting\_time += currentTime - processes[j].arrival\_time;  
            }  
        }  
    }  
}  
  
void calculateTurnaroundTime(struct Process processes[], int n) {  
    int i;  
  
    for (i = 0; i < n; ++i) {  
        processes[i].turnaround\_time = processes[i].burst\_time + processes[i].waiting\_time;  
    }  
}  
  
void calculateCompletionTime(struct Process processes[], int n) {  
    int i;  
  
    for (i = 0; i < n; ++i) {  
        processes[i].completion\_time = processes[i].start\_time + processes[i].turnaround\_time;  
    }  
}  
  
void calculatePreemptivePriority(struct Process processes[], int n) {  
    int currentTime = 0, executed = 0, i, j, nextProcess;  
  
    while (executed < n) {  
        nextProcess = -1;  
  
        for (i = 0; i < n; ++i) {  
            if (processes[i].arrival\_time <= currentTime && processes[i].remaining\_time > 0) {  
                if (nextProcess == -1) {  
                    nextProcess = i;  
                } else if (processes[i].priority < processes[nextProcess].priority) {  
                    nextProcess = i;  
                }  
            }  
        }  
  
        if (nextProcess == -1) {  
            currentTime++;  
        } else {  
            if (processes[nextProcess].start\_time == -1) {  
                processes[nextProcess].start\_time = currentTime;  
            }  
  
            processes[nextProcess].remaining\_time--;  
            currentTime++;  
            processes[nextProcess].executed\_time++;  
  
            if (processes[nextProcess].remaining\_time == 0) {  
                executed++;  
                processes[nextProcess].completion\_time = currentTime;  
            }  
  
            for (j = 0; j < n; ++j) {  
                if (j != nextProcess && processes[j].arrival\_time <= currentTime && processes[j].remaining\_time > 0) {  
                    processes[j].waiting\_time++;  
                }  
            }  
        }  
    }  
}  
  
void displayProcessDetails(struct Process processes[], int n) {  
    int i;  
  
    printf("\nProcess ID\tPriority\tBurst Time\tArrival Time\tStart Time\tCompletion Time\tWaiting Time\tTurnaround Time\n");  
  
    for (i = 0; i < n; ++i) {  
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t%d\n", processes[i].pid, processes[i].priority, processes[i].burst\_time, processes[i].arrival\_time, processes[i].start\_time, processes[i].completion\_time, processes[i].waiting\_time, processes[i].turnaround\_time);  
    }  
}  
  
int main() {  
    int n, i;  
  
    printf("Enter the number of processes: ");  
    scanf("%d", &n);  
  
    struct Process processes[n];  
  
    printf("\nEnter process details:\n");  
  
    for (i = 0; i < n; ++i) {  
        processes[i].pid = i + 1;  
        printf("Enter priority for process %d: ", i + 1);  
        scanf("%d", &processes[i].priority);  
        printf("Enter burst time for process %d: ", i + 1);  
        scanf("%d", &processes[i].burst\_time);  
        printf("Enter arrival time for process %d: ", i + 1);  
        scanf("%d", &processes[i].arrival\_time);  
        processes[i].start\_time = -1;  
        processes[i].completion\_time = -1;  
        processes[i].waiting\_time = 0;  
        processes[i].turnaround\_time = 0;  
        processes[i].remaining\_time = processes[i].burst\_time;  
        processes[i].executed\_time = 0;  
    }  
  
    calculatePreemptivePriority(processes, n);  
    calculateWaitingTime(processes, n);  
    calculateTurnaroundTime(processes, n);  
    calculateCompletionTime(processes, n);  
  
    displayProcessDetails(processes, n);  
    displayGanttChart(processes, n);  
  
    return 0;  
}