#include <stdio.h>

#include <stdlib.h>

struct Process {

int pid;

int burstTime;

int priority;

int waitingTime;

int turnaroundTime;

};

void sortByPriority(struct Process\* processes, int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (processes[j].priority > processes[j + 1].priority) {

// Swap the processes

struct Process temp = processes[j];

processes[j] = processes[j + 1];

processes[j + 1] = temp;

}

}

}

}

void calculateWaitingTime(struct Process\* processes, int n) {

processes[0].waitingTime = 0;

for (int i = 1; i < n; i++) {

processes[i].waitingTime = processes[i - 1].waitingTime + processes[i - 1].burstTime;

}

}

void calculateTurnaroundTime(struct Process\* processes, int n) {

for (int i = 0; i < n; i++) {

processes[i].turnaroundTime = processes[i].waitingTime + processes[i].burstTime;

}

}

void displayGanttChart(struct Process\* processes, int n) {

printf("\nGantt Chart:\n");

for (int i = 0; i < n; i++) {

printf("|\tP%d\t", processes[i].pid);

}

printf("|\n");

for (int i = 0; i < n; i++) {

printf("%d\t\t", processes[i].waitingTime);

}

printf("%d\n", processes[n - 1].waitingTime + processes[n - 1].burstTime);

}

void displayProcessTable(struct Process\* processes, int n) {

printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");

for (int i = 0; i < n; i++) {

printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burstTime, processes[i].priority, processes[i].waitingTime, processes[i].turnaroundTime);

}

}

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process\* processes = (struct Process\*)malloc(n \* sizeof(struct Process));

for (int i = 0; i < n; i++) {

printf("Enter burst time and priority for process P%d: ", i);

scanf("%d %d", &processes[i].burstTime, &processes[i].priority);

processes[i].pid = i;

}

sortByPriority(processes, n);

calculateWaitingTime(processes, n);

calculateTurnaroundTime(processes, n);

displayGanttChart(processes, n);

displayProcessTable(processes, n);

free(processes);

return 0;

}