

NEW RR

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    char pname[20];
    int at, bt, ct, tat, wt;
    int bt1;
} Process;

typedef struct {
    int start;
    char pname[20];
    int end;
} Gantt;

Process processes[10];
Gantt ganttChart[100];
int processCount, timeQuantum, ganttIndex = 0;

void addToGanttChart(int start, int end, char pname[]) {
    ganttChart[ganttIndex].start = start;
    ganttChart[ganttIndex].end = end;
    strcpy(ganttChart[ganttIndex].pname, pname);
    ganttIndex++;
}

void printGanttChart() {
    printf("\nGantt Chart:\n");
    for (int i = 0; i < ganttIndex; i++) {
        printf("| %s ", ganttChart[i].pname);
    }
    printf("\n");

    for (int i = 0; i < ganttIndex; i++) {
        printf("%d ", ganttChart[i].start);
    }
}
```

```

    printf("%d\n", ganttChart[ganttIndex - 1].end);
}

```

```

void acceptProcessInfo() {
    printf("Enter the number of processes: ");
    scanf("%d", &processCount);
    for (int i = 0; i < processCount; i++) {
        printf("Enter process name: ");
        scanf("%s", processes[i].pname);
        printf("Enter arrival time: ");
        scanf("%d", &processes[i].at);
        printf("Enter burst time: ");
        scanf("%d", &processes[i].bt);
        processes[i]×bt1 = processes[i].bt;
    }
    printf("Enter time quantum: ");
    scanf("%d", &timeQuantum);
}

```

```

void roundRobinScheduling() {
    int time = 0, completed = 0;
    while (completed < processCount) {
        for (int i = 0; i < processCount; i++) {
            if (processes[i].bt1 > 0 && processes[i].at <= time) {
                int execTime = (processes[i].bt1 > timeQuantum) ? timeQuantum :
processes[i].bt1;
                addToGanttChart(time, time + execTime, processes[i].pname);
                time += execTime;
                processes[i]×bt1 -= execTime;

                if (processes[i]×bt1 == 0) {
                    processes[i]×ct = time;
                    completed++;
                }
            }
        }
    }
}

```

```

void calculateTATandWT() {
    for (int i = 0; i < processCount; i++) {
        processes[i]×tat = processes[i].ct - processes[i].at;
        processes[i]×wt = processes[i].tat - processes[i].bt;
    }
}

```

```
    }  
}
```

```
void displayResults() {  
    printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");  
    for (int i = 0; i < processCount; i++) {  
        printf("%s\t%d\t%d\t%d\t%d\t%d\n",  
            processes[i].pname, processes[i].at, processes[i].bt,  
            processes[i].ct, processes[i].tat, processes[i].wt);  
    }  
}
```

```
void calculateAndPrintAverages() {  
    float totalTAT = 0, totalWT = 0;  
    for (int i = 0; i < processCount; i++) {  
        totalTAT += processes[i].tat;  
        totalWT += processes[i].wt;  
    }  
    printf("Average Turnaround Time: %.2f\n", totalTAT / processCount);  
    printf("Average Waiting Time: %.2f\n", totalWT / processCount);  
}
```

```
int main() {  
    acceptProcessInfo();  
    roundRobinScheduling();  
    calculateTATandWT();  
    displayResults();  
    printGanttChart(); // Print Gantt chart  
    calculateAndPrintAverages();  
    return 0;  
}
```