ROUND ROBIN

```
#include <stdio.h>

struct Process {
    int pid, arrival, burst, remaining, completion, waiting, turnaround;
};

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

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("\nEnter Arrival Time for P%d: ", i + 1);
        scanf("%d", &p[i].arrival);
        printf("Enter Burst Time for P%d: ", i + 1);
        scanf("%d", &p[i].burst);
        p[i].remaining = p[i].burst;
    }

    printf("\nEnter Time Quantum: ");
    scanf("%d", &time_quantum);

    int current_time = 0, completed = 0;
    float total_wait = 0, total_turnaround = 0;
    int queue[100], front = 0, rear = 0;
    int visited[n];
    for (int i = 0; i < n; i++) visited[i] = 0;
```

```
queue[rear++] = 0;
visited[0] = 1;
current_time = p[0].arrival;

printf("\n--- Round Robin Scheduling ---\n");

while (completed < n) {
    int i = queue[front++];
    if (p[i].remaining > time_quantum) {
        p[i].remaining -= time_quantum;
        current_time += time_quantum;
    } else {
        current_time += p[i].remaining;
        p[i].remaining = 0;
        completed++;
        p[i].completion = current_time;
        p[i].turnaround = p[i].completion - p[i].arrival;
        p[i].waiting = p[i].turnaround - p[i].burst;
        total_wait += p[i].waiting;
        total_turnaround += p[i].turnaround;
    }

    for (int j = 0; j < n; j++) {
        if (p[j].arrival <= current_time && p[j].remaining > 0 && !visited[j]) {
            queue[rear++] = j;
            visited[j] = 1;
        }
    }

    if (p[i].remaining > 0)
```

```
            queue[rear++] = i;

        if (front == rear && completed < n) {
            for (int j = 0; j < n; j++) {
                if (p[j].remaining > 0) {
                    queue[rear++] = j;
                    visited[j] = 1;
                    current_time = p[j].arrival;
                    break;
                }
            }
        }
    }

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival, p[i].burst,
               p[i].completion, p[i].turnaround, p[i].waiting);
    }

    printf("\nAverage Turnaround Time = %.2f", total_turnaround / n);
    printf("\nAverage Waiting Time = %.2f\n", total_wait / n);

    return 0;
}
```

Enter number of processes: 3

Enter Arrival Time for P1: 0

Enter Burst Time for P1: 5

Enter Arrival Time for P2: 1

Enter Burst Time for P2: 4

Enter Arrival Time for P3: 2

Enter Burst Time for P3: 2

Enter Time Quantum: 2

--- Round Robin Scheduling ---

PID	AT	BT	CT	TAT	WT
P1	0	5	11	11	6
P2	1	4	10	9	5
P3	2	2	6	4	2

Average Turnaround Time = 8.00

Average Waiting Time = 4.33

example

Part B — Round Robin (Different Arrival Times)

We'll take the same processes again:

Process	Arrival Time	Burst Time
P1	0	5
P2	1	4
P3	2	2

and **Time Quantum = 2**.

Step 1: Gantt Chart Simulation

Processes at start:

- t=0 → only P1 is available, so start P1.

Time	Running	Ready Queue (after execution)
0–2	P1 (BT 5→3)	[P2, P3, P1]
2–4	P2 (BT 4→2)	[P3, P1, P2]
4–6	P3 (BT 2→0)	[P1, P2] (P3 finishes at 6)
6–8	P1 (BT 3→1)	[P2, P1]
8–10	P2 (BT 2→0)	[P1] (P2 finishes at 10)
10–11	P1 (BT 1→0)	[] (P1 finishes at 11)

✅ **Completion Times (CT):**

- P1 = 11
- P2 = 10
- P3 = 6

Step 2: Calculate TAT & WT

Process	AT	BT	CT	TAT = CT−AT	WT = TAT−BT
P1	0	5	11	11	6
P2	1	4	10	9	5
P3	2	2	6	4	2

Step 3: Average Times

- **Average TAT = (11 + 9 + 4)/3 = 8.00**

- **Average WT = (6 + 5 + 2)/3 = 4.33**

✅ **Final Paper Answer (Round Robin)**

PID AT BT CT TAT WT

P1 0 5 11 11 6

P2 1 4 10 9 5

P3 2 2 6 4 2

Average TAT = 8.00
Average WT = 4.33