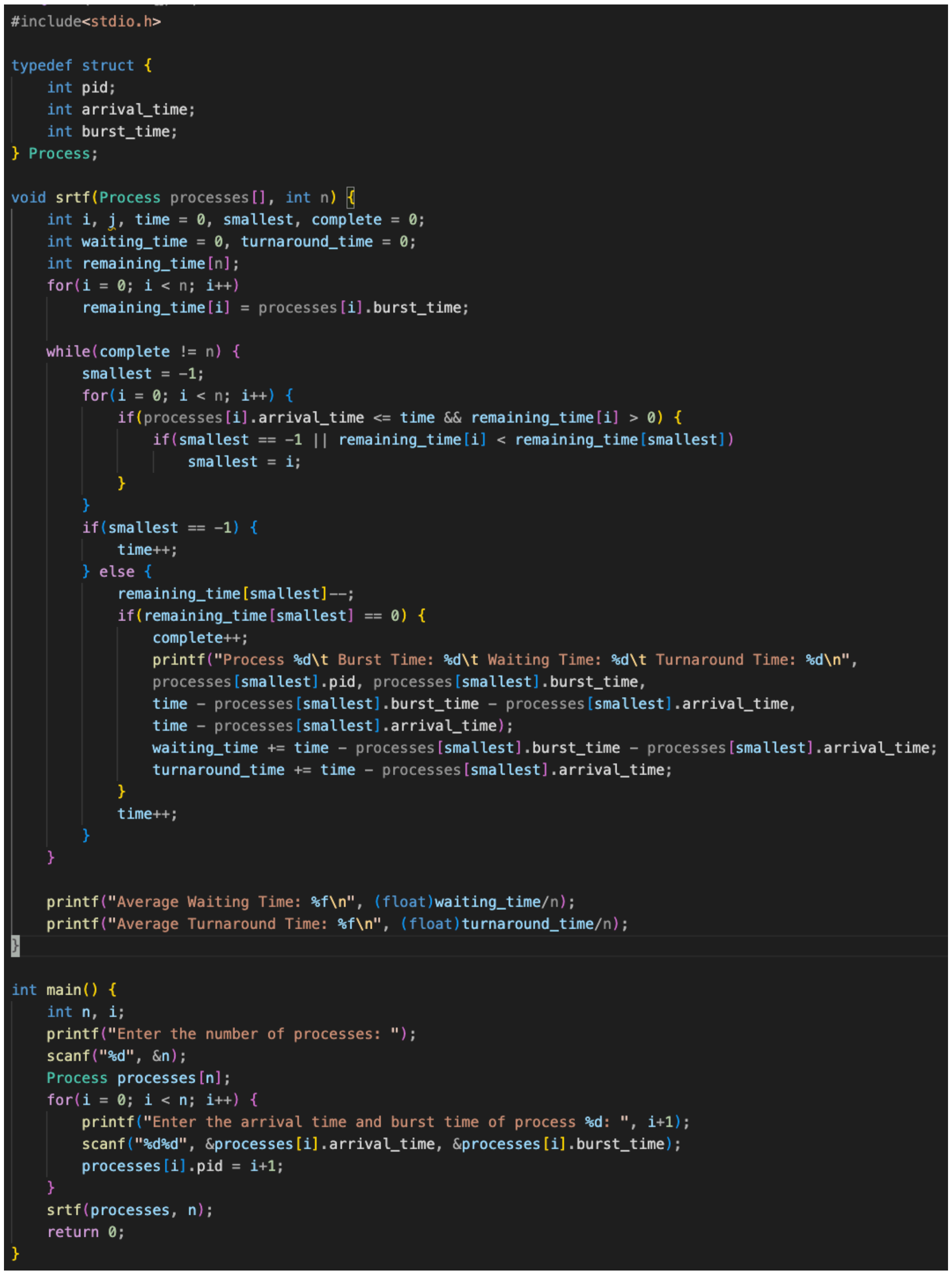
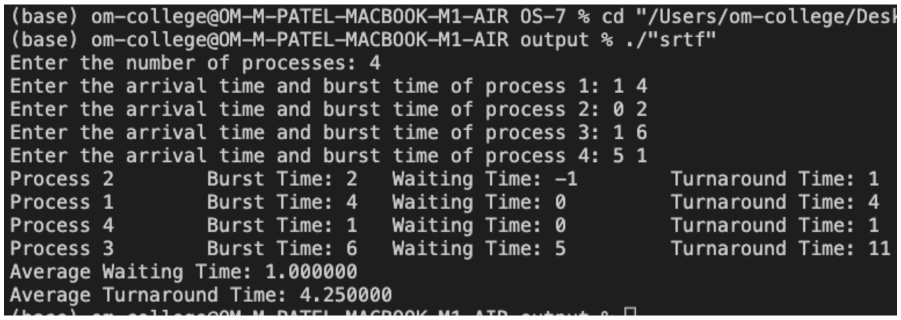
EXPERIMENT -7 :SCHEDULING ALGORITHM

3) Shortest remaining time first:- It is the preemptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution.



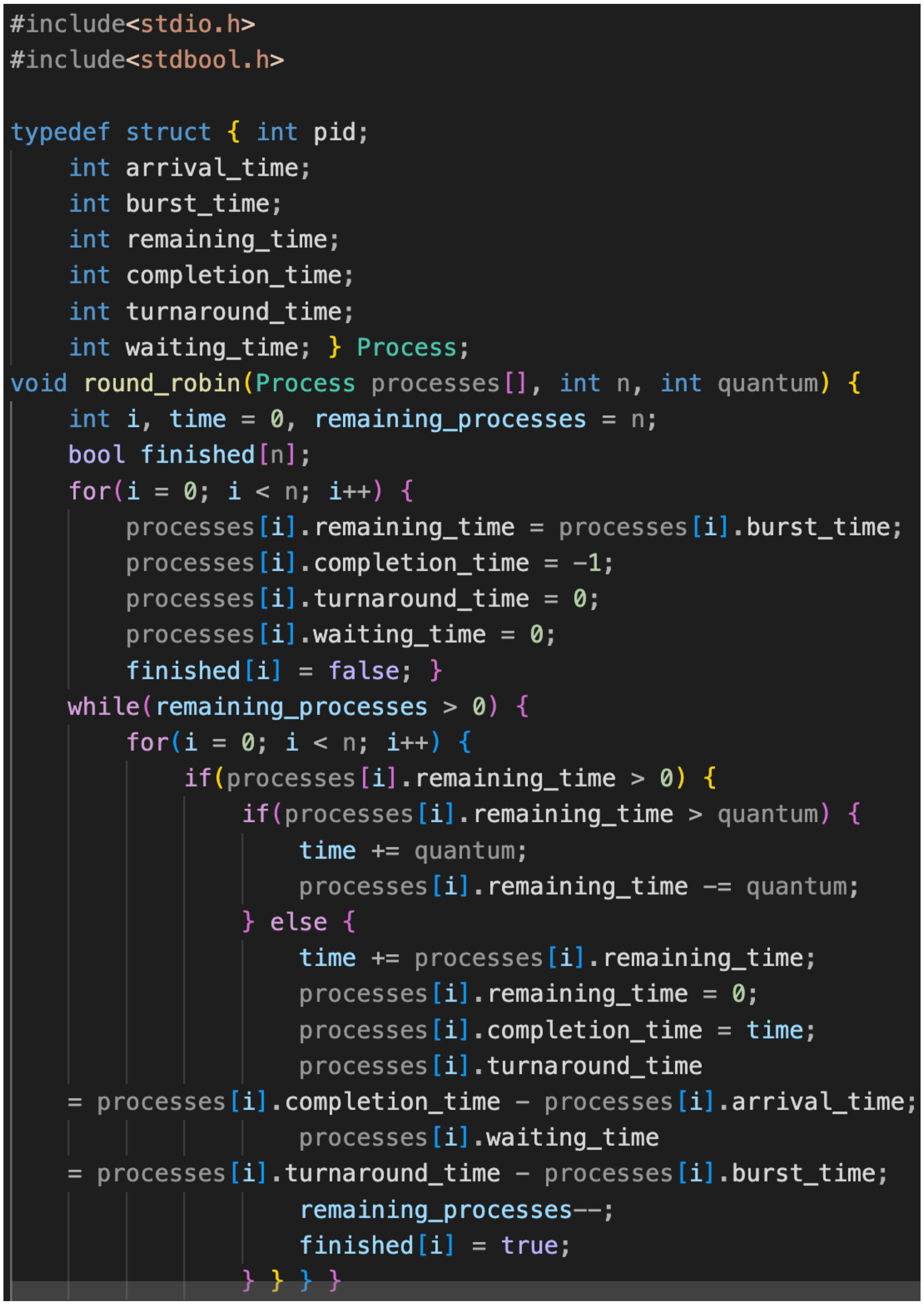
```c
#include<stdio.h>

typedef struct {
    int pid;
    int arrival_time;
    int burst_time;
} Process;

void srtf(Process processes[], int n) {
    int i, j, time = 0, smallest, complete = 0;
    int waiting_time = 0, turnaround_time = 0;
    int remaining_time[n];
    for(i = 0; i < n; i++)
        remaining_time[i] = processes[i].burst_time;

    while(complete != n) {
        smallest = -1;
        for(i = 0; i < n; i++) {
            if(processes[i].arrival_time <= time && remaining_time[i] > 0) {
                if(smallest == -1 || remaining_time[i] < remaining_time[smallest])
                    smallest = i;
            }
        }
        if(smallest == -1) {
            time++;
        } else {
            remaining_time[smallest]--;
            if(remaining_time[smallest] == 0) {
                complete++;
                printf("Process %d\t Burst Time: %d\t Waiting Time: %d\t Turnaround Time: %d\n",
                processes[smallest].pid, processes[smallest].burst_time,
                time - processes[smallest].burst_time - processes[smallest].arrival_time,
                time - processes[smallest].arrival_time);
                waiting_time += time - processes[smallest].burst_time - processes[smallest].arrival_time;
                turnaround_time += time - processes[smallest].arrival_time;
            }
            time++;
        }
    }

    printf("Average Waiting Time: %f\n", (float)waiting_time/n);
    printf("Average Turnaround Time: %f\n", (float)turnaround_time/n);
}

int main() {
    int n, i;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    Process processes[n];
    for(i = 0; i < n; i++) {
        printf("Enter the arrival time and burst time of process %d: ", i+1);
        scanf("%d%d", &processes[i].arrival_time, &processes[i].burst_time);
        processes[i].pid = i+1;
    }
    srtf(processes, n);
    return 0;
}
```



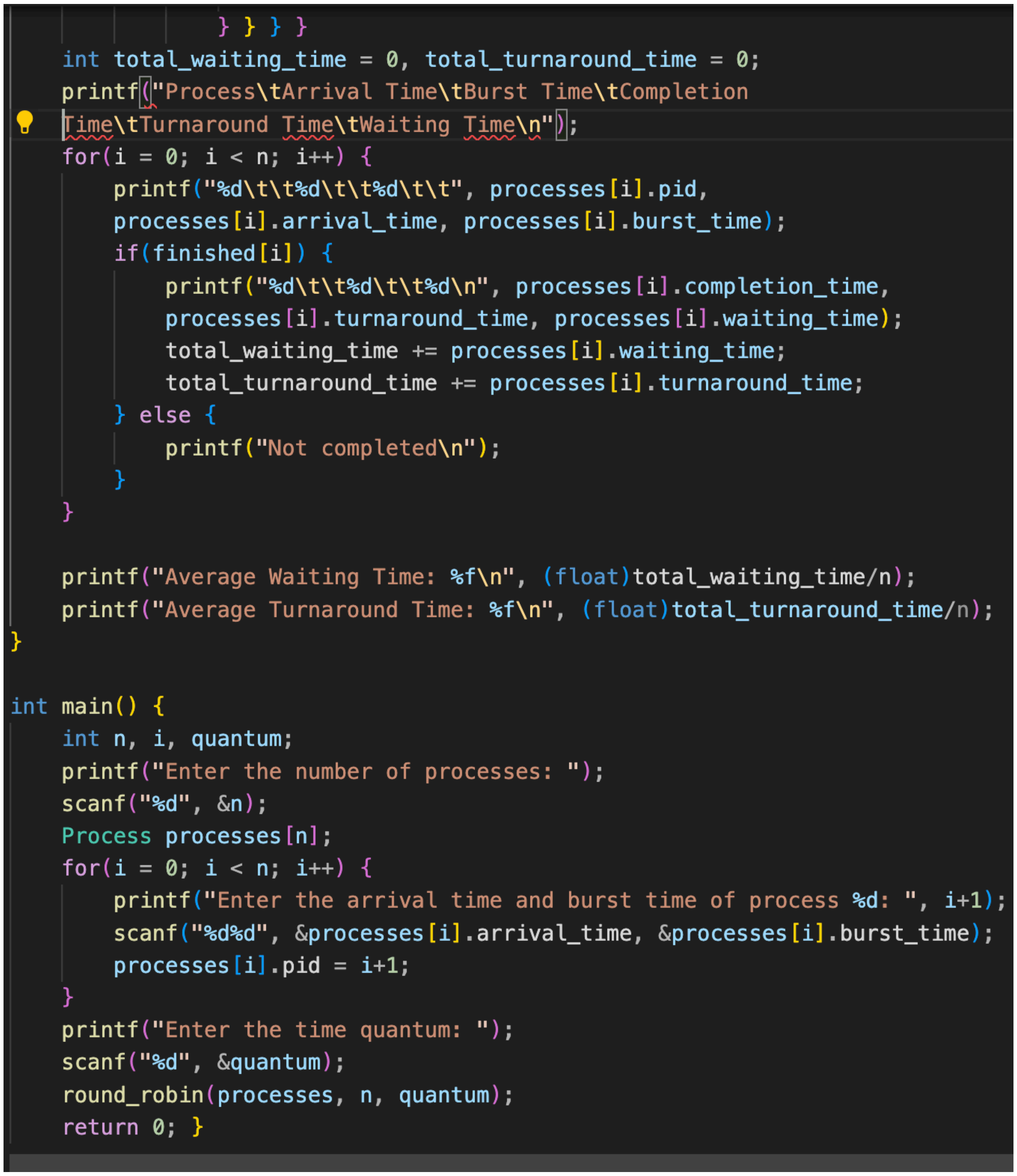
```
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR OS-7 % cd "/Users/om-college/Desk
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR output % ./"srtf"
Enter the number of processes: 4
Enter the arrival time and burst time of process 1: 1 4
Enter the arrival time and burst time of process 2: 0 2
Enter the arrival time and burst time of process 3: 1 6
Enter the arrival time and burst time of process 4: 5 1
Process 2        Burst Time: 2   Waiting Time: -1        Turnaround Time: 1
Process 1        Burst Time: 4   Waiting Time: 0         Turnaround Time: 4
Process 4        Burst Time: 1   Waiting Time: 0         Turnaround Time: 1
Process 3        Burst Time: 6   Waiting Time: 5         Turnaround Time: 11
Average Waiting Time: 1.000000
Average Turnaround Time: 4.250000
```

1) Round Robin:-

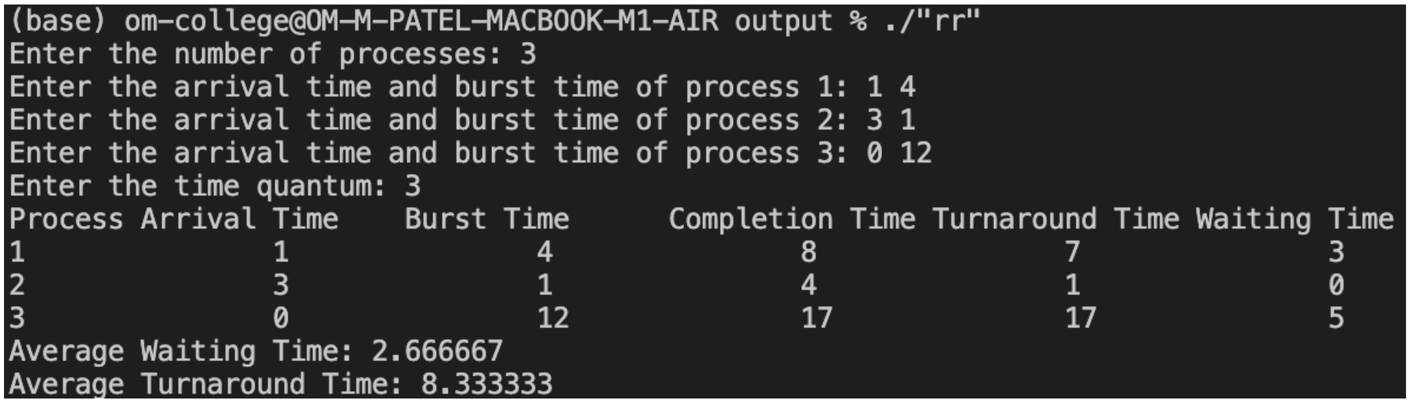
In the Round Robin scheduling algorithm, the OS defines a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a preemptive type of scheduling.



```
#include<stdio.h>
#include<stdbool.h>

typedef struct { int pid;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int completion_time;
    int turnaround_time;
    int waiting_time; } Process;
void round_robin(Process processes[], int n, int quantum) {
    int i, time = 0, remaining_processes = n;
    bool finished[n];
    for(i = 0; i < n; i++) {
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].completion_time = -1;
        processes[i].turnaround_time = 0;
        processes[i].waiting_time = 0;
        finished[i] = false; }
    while(remaining_processes > 0) {
        for(i = 0; i < n; i++) {
            if(processes[i].remaining_time > 0) {
                if(processes[i].remaining_time > quantum) {
                    time += quantum;
                    processes[i].remaining_time -= quantum;
                } else {
                    time += processes[i].remaining_time;
                    processes[i].remaining_time = 0;
                    processes[i].completion_time = time;
                    processes[i].turnaround_time
    = processes[i].completion_time - processes[i].arrival_time;
                    processes[i].waiting_time
    = processes[i].turnaround_time - processes[i].burst_time;
                    remaining_processes--;
                    finished[i] = true;
                } } } }
```



```c
                } } } }
    int total_waiting_time = 0, total_turnaround_time = 0;
    printf("Process\tArrival Time\tBurst Time\tCompletion
Time\tTurnaround Time\tWaiting Time\n");
    for(i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t", processes[i].pid,
        processes[i].arrival_time, processes[i].burst_time);
        if(finished[i]) {
            printf("%d\t\t%d\t\t%d\n", processes[i].completion_time,
            processes[i].turnaround_time, processes[i].waiting_time);
            total_waiting_time += processes[i].waiting_time;
            total_turnaround_time += processes[i].turnaround_time;
        } else {
            printf("Not completed\n");
        }
    }

    printf("Average Waiting Time: %f\n", (float)total_waiting_time/n);
    printf("Average Turnaround Time: %f\n", (float)total_turnaround_time/n);
}

int main() {
    int n, i, quantum;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    Process processes[n];
    for(i = 0; i < n; i++) {
        printf("Enter the arrival time and burst time of process %d: ", i+1);
        scanf("%d%d", &processes[i].arrival_time, &processes[i].burst_time);
        processes[i].pid = i+1;
    }
    printf("Enter the time quantum: ");
    scanf("%d", &quantum);
    round_robin(processes, n, quantum);
    return 0; }
```



```
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR output % ./"rr"
Enter the number of processes: 3
Enter the arrival time and burst time of process 1: 1 4
Enter the arrival time and burst time of process 2: 3 1
Enter the arrival time and burst time of process 3: 0 12
Enter the time quantum: 3
Process Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
1               1               4               8               7               3
2               3               1               4               1               0
3               0               12              17              17              5
Average Waiting Time: 2.666667
Average Turnaround Time: 8.333333
```