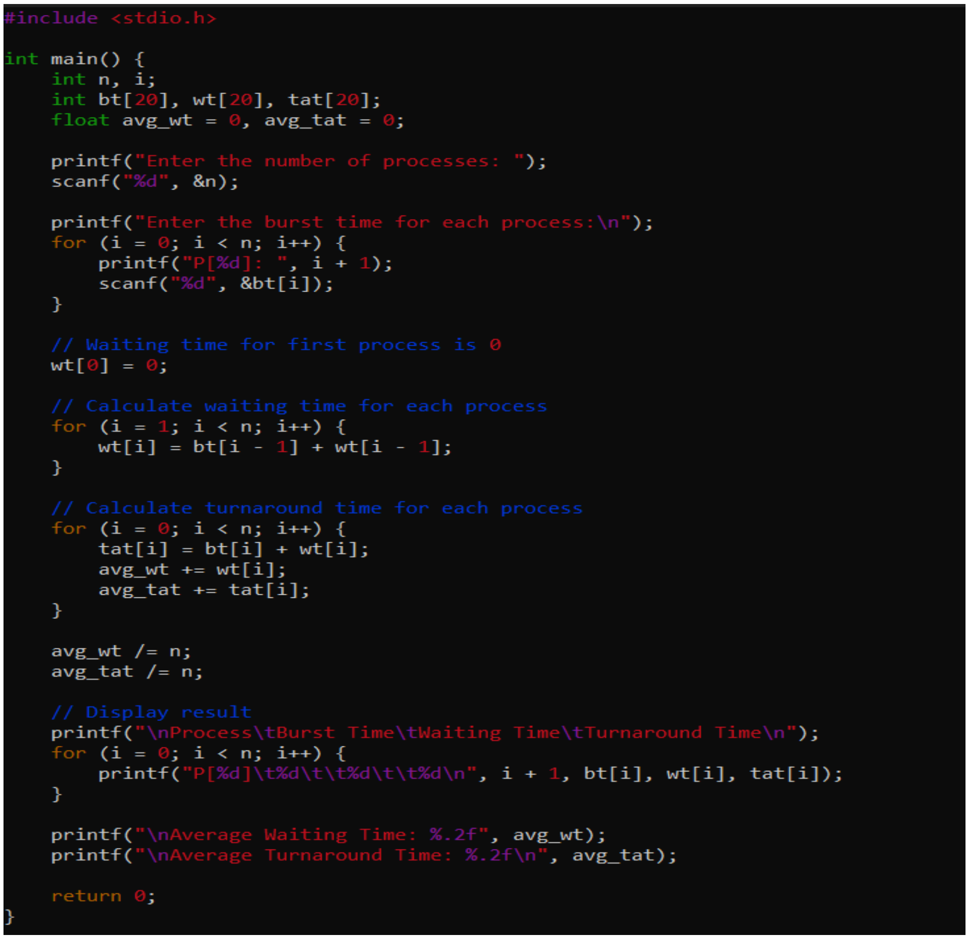
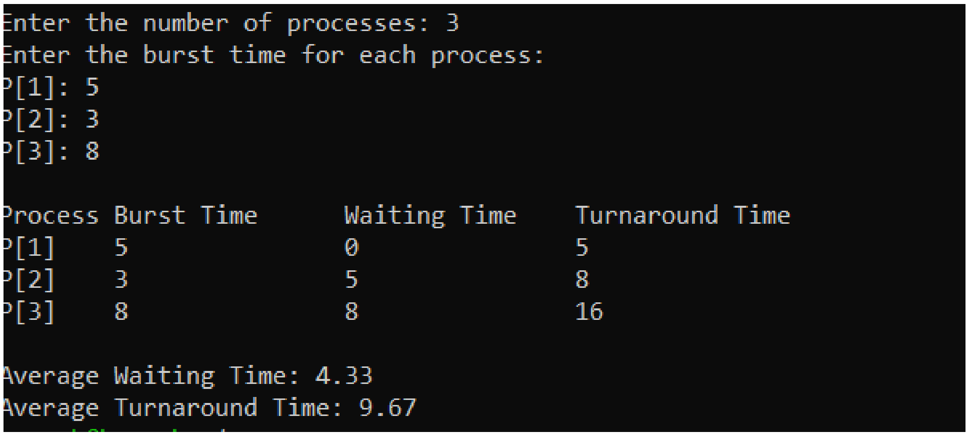
Ex. No.6a) FIRST COME FIRST SERVE

Program code:



```c
#include <stdio.h>

int main() {
    int n, i;
    int bt[20], wt[20], tat[20];
    float avg_wt = 0, avg_tat = 0;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter the burst time for each process:\n");
    for (i = 0; i < n; i++) {
        printf("P[%d]: ", i + 1);
        scanf("%d", &bt[i]);
    }

    // Waiting time for first process is 0
    wt[0] = 0;

    // Calculate waiting time for each process
    for (i = 1; i < n; i++) {
        wt[i] = bt[i - 1] + wt[i - 1];
    }

    // Calculate turnaround time for each process
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avg_wt += wt[i];
        avg_tat += tat[i];
    }

    avg_wt /= n;
    avg_tat /= n;

    // Display result
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("P[%d]\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

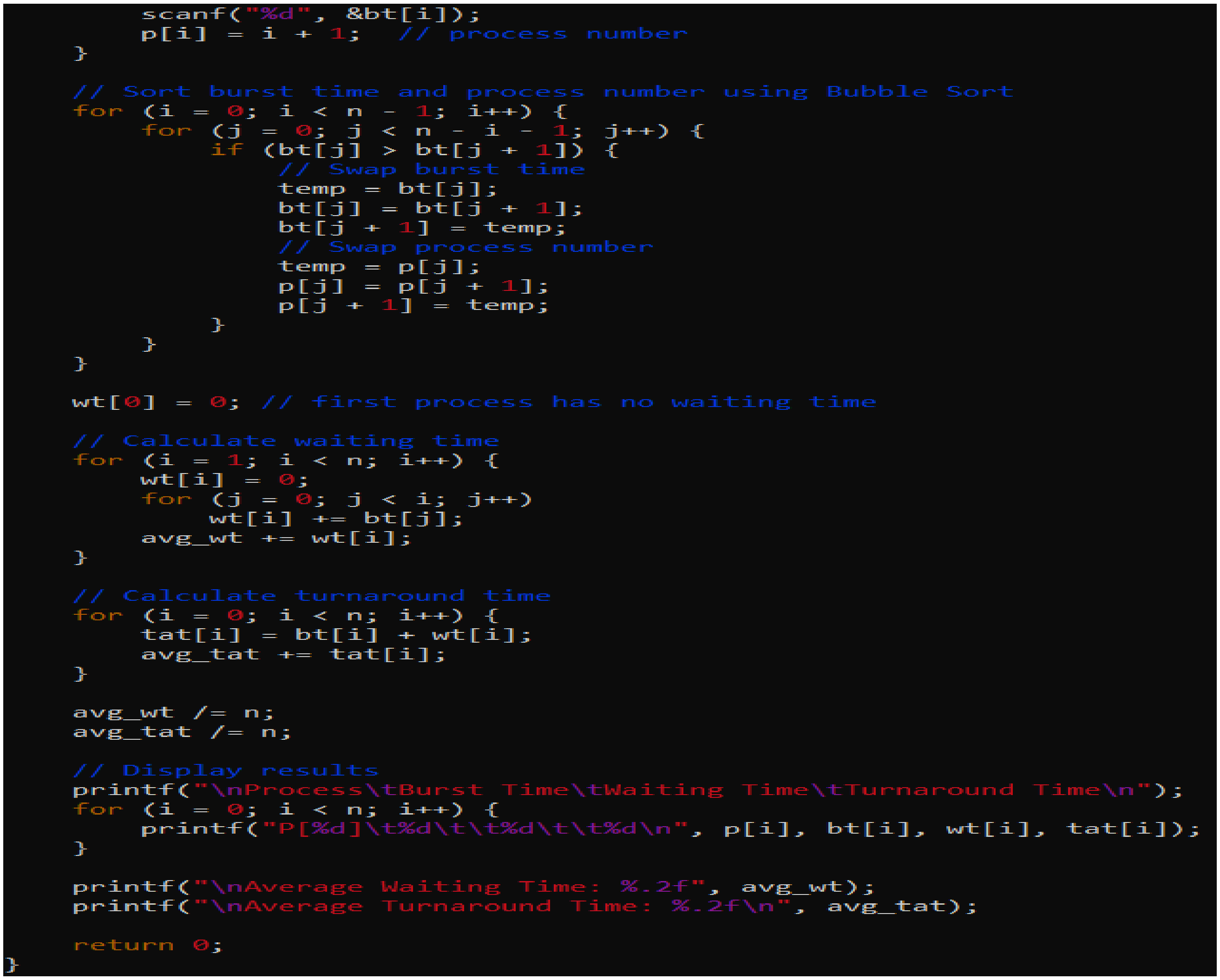
Output:



```
Enter the number of processes: 3
Enter the burst time for each process:
P[1]: 5
P[2]: 3
P[3]: 8

Process Burst Time      Waiting Time    Turnaround Time
P[1]    5               0               5
P[2]    3               5               8
P[3]    8               8               16

Average Waiting Time: 4.33
Average Turnaround Time: 9.67
```

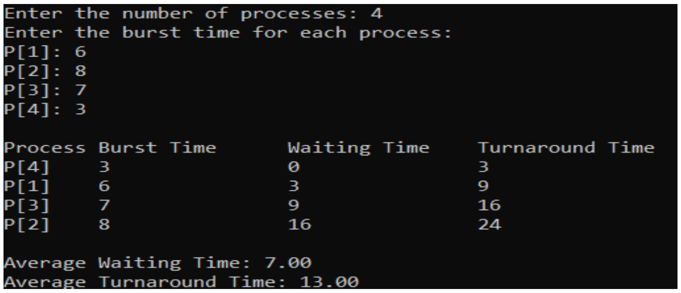
6b) To implement the Shortest Job First (SJF) scheduling technique

Program code:



```c
        scanf("%d", &bt[i]);
        p[i] = i + 1;  // process number
    }

    // Sort burst time and process number using Bubble Sort
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (bt[j] > bt[j + 1]) {
                // Swap burst time
                temp = bt[j];
                bt[j] = bt[j + 1];
                bt[j + 1] = temp;
                // Swap process number
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }

    wt[0] = 0; // first process has no waiting time

    // Calculate waiting time
    for (i = 1; i < n; i++) {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];
        avg_wt += wt[i];
    }

    // Calculate turnaround time
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avg_tat += tat[i];
    }

    avg_wt /= n;
    avg_tat /= n;

    // Display results
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("P[%d]\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

Output:



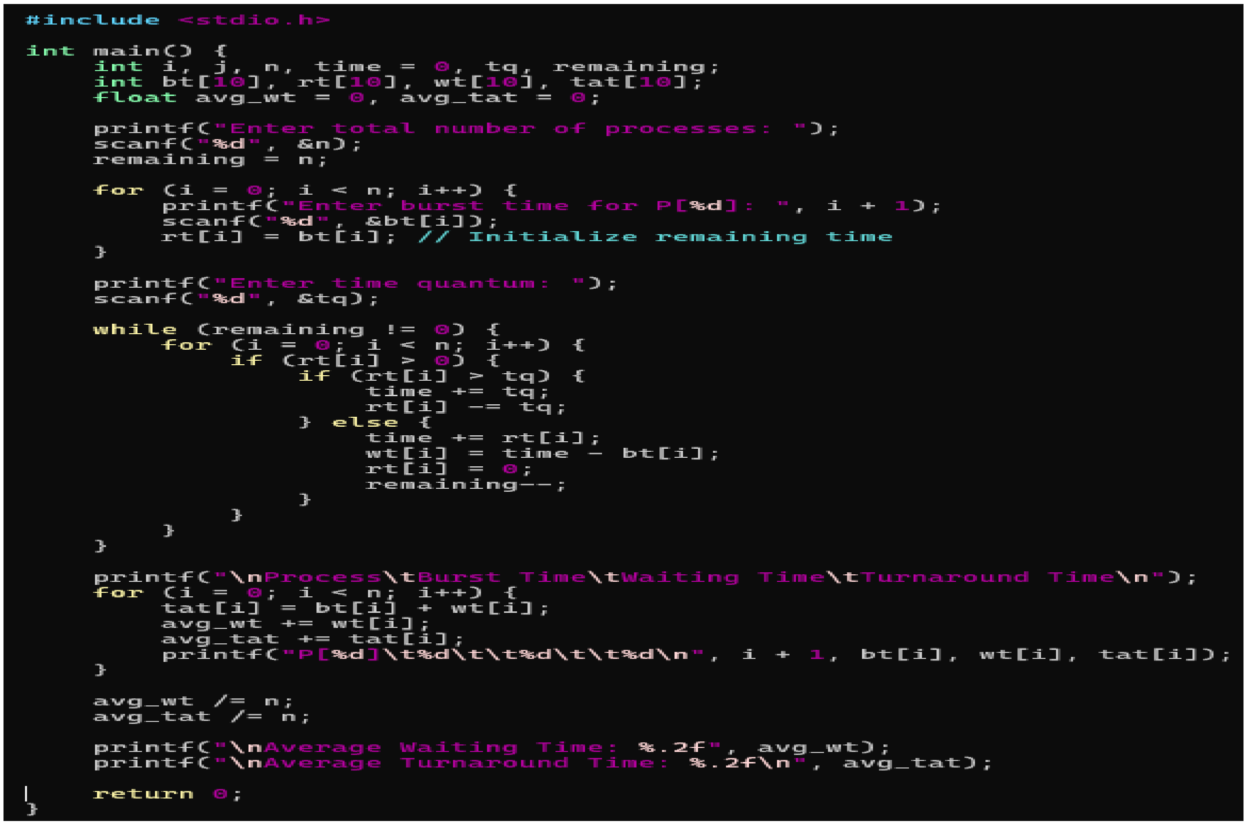
```
Enter the number of processes: 4
Enter the burst time for each process:
P[1]: 6
P[2]: 8
P[3]: 7
P[4]: 3

Process Burst Time      Waiting Time    Turnaround Time
P[4]    3               0               3
P[1]    6               3               9
P[3]    7               9               16
P[2]    8               16              24

Average Waiting Time: 7.00
Average Turnaround Time: 13.00
```

Ex. No.: 6d)

ROUND ROBIN SCHEDULING

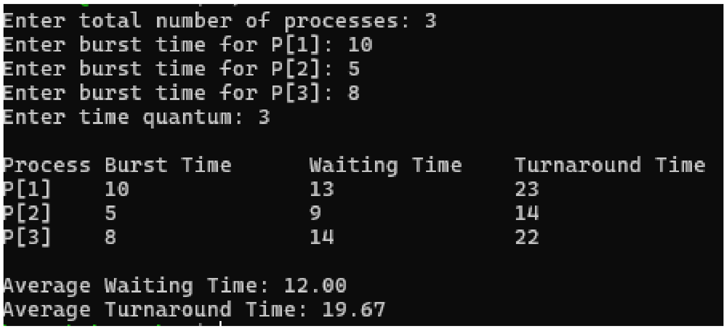
Aim: To implement the Round Robin (RR) scheduling technique

Program code:



```c
#include <stdio.h>

int main() {
    int i, j, n, time = 0, tq, remaining;
    int bt[10], rt[10], wt[10], tat[10];
    float avg_wt = 0, avg_tat = 0;

    printf("Enter total number of processes: ");
    scanf("%d", &n);
    remaining = n;

    for (i = 0; i < n; i++) {
        printf("Enter burst time for P[%d]: ", i + 1);
        scanf("%d", &bt[i]);
        rt[i] = bt[i]; // Initialize remaining time
    }

    printf("Enter time quantum: ");
    scanf("%d", &tq);

    while (remaining != 0) {
        for (i = 0; i < n; i++) {
            if (rt[i] > 0) {
                if (rt[i] > tq) {
                    time += tq;
                    rt[i] -= tq;
                } else {
                    time += rt[i];
                    wt[i] = time - bt[i];
                    rt[i] = 0;
                    remaining--;
                }
            }
        }
    }

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avg_wt += wt[i];
        avg_tat += tat[i];
        printf("P[%d]\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }

    avg_wt /= n;
    avg_tat /= n;

    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

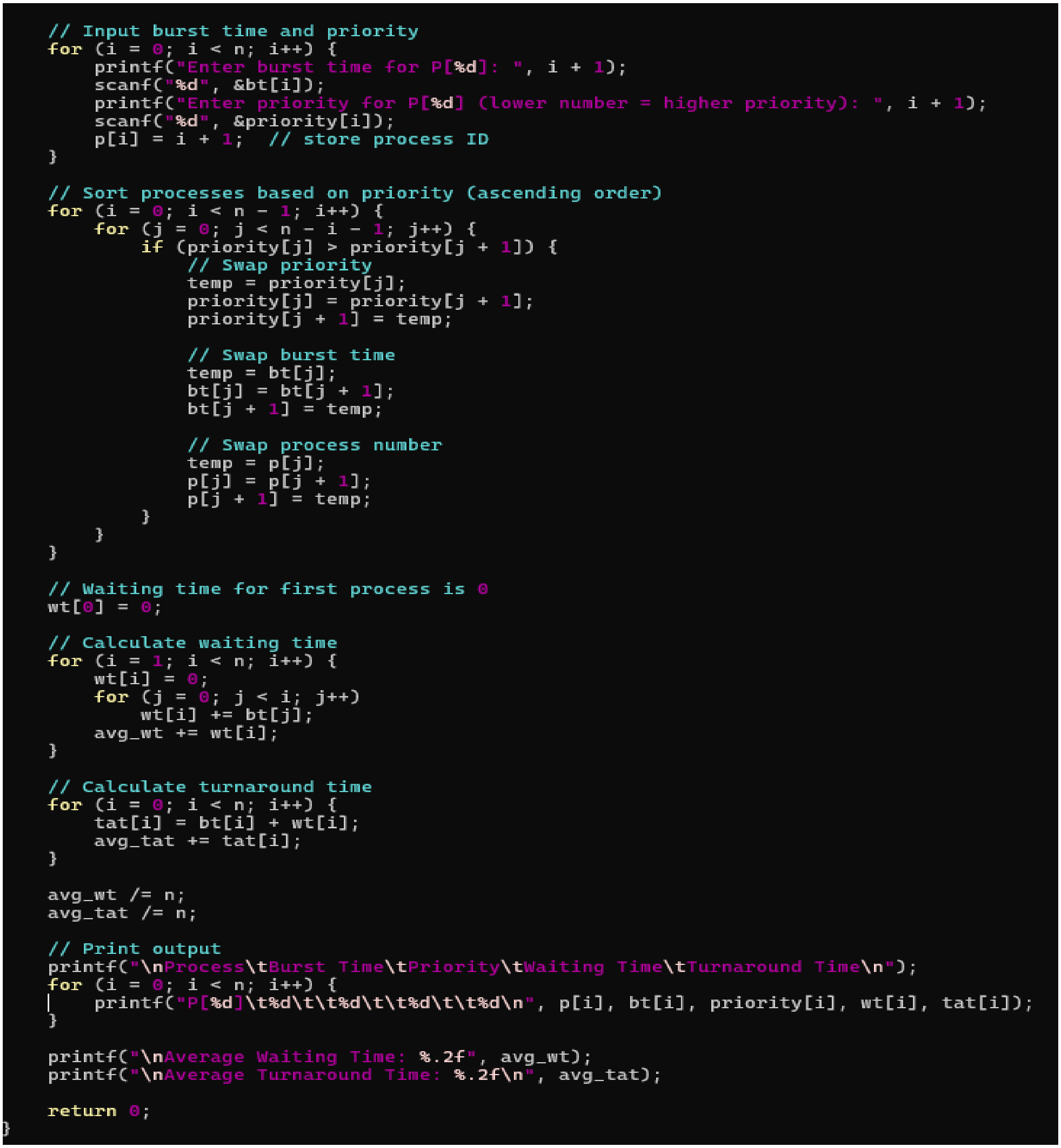
Output:



```
Enter total number of processes: 3
Enter burst time for P[1]: 10
Enter burst time for P[2]: 5
Enter burst time for P[3]: 8
Enter time quantum: 3

Process Burst Time      Waiting Time    Turnaround Time
P[1]    10              13              23
P[2]    5               9               14
P[3]    8               14              22

Average Waiting Time: 12.00
Average Turnaround Time: 19.67
```

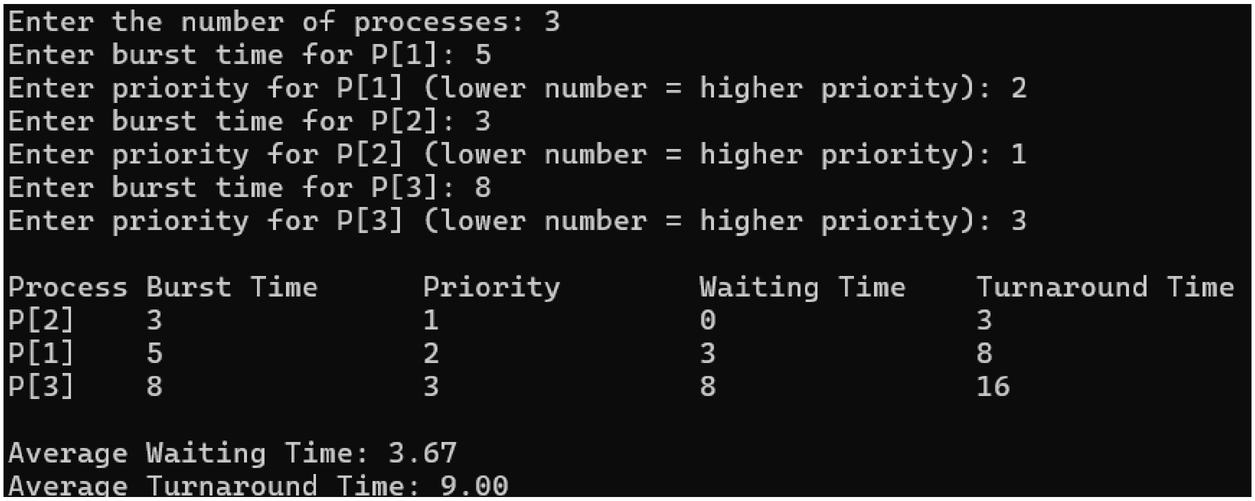
Ex. No. 6c) PRIORITY SCHEDULING

Progam Code:



```
    // Input burst time and priority
    for (i = 0; i < n; i++) {
        printf("Enter burst time for P[%d]: ", i + 1);
        scanf("%d", &bt[i]);
        printf("Enter priority for P[%d] (lower number = higher priority): ", i + 1);
        scanf("%d", &priority[i]);
        p[i] = i + 1;  // store process ID
    }

    // Sort processes based on priority (ascending order)
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (priority[j] > priority[j + 1]) {
                // Swap priority
                temp = priority[j];
                priority[j] = priority[j + 1];
                priority[j + 1] = temp;

                // Swap burst time
                temp = bt[j];
                bt[j] = bt[j + 1];
                bt[j + 1] = temp;

                // Swap process number
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }

    // Waiting time for first process is 0
    wt[0] = 0;

    // Calculate waiting time
    for (i = 1; i < n; i++) {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];
        avg_wt += wt[i];
    }

    // Calculate turnaround time
    for (i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        avg_tat += tat[i];
    }

    avg_wt /= n;
    avg_tat /= n;

    // Print output
    printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("P[%d]\t%d\t\t%d\t\t%d\t\t%d\n", p[i], bt[i], priority[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", avg_wt);
    printf("\nAverage Turnaround Time: %.2f\n", avg_tat);

    return 0;
}
```

Output (6c Priority Scheduling):



```
Enter the number of processes: 3
Enter burst time for P[1]: 5
Enter priority for P[1] (lower number = higher priority): 2
Enter burst time for P[2]: 3
Enter priority for P[2] (lower number = higher priority): 1
Enter burst time for P[3]: 8
Enter priority for P[3] (lower number = higher priority): 3

Process Burst Time      Priority        Waiting Time    Turnaround Time
P[2]    3               1               0               3
P[1]    5               2               3               8
P[3]    8               3               8               16

Average Waiting Time: 3.67
Average Turnaround Time: 9.00
```