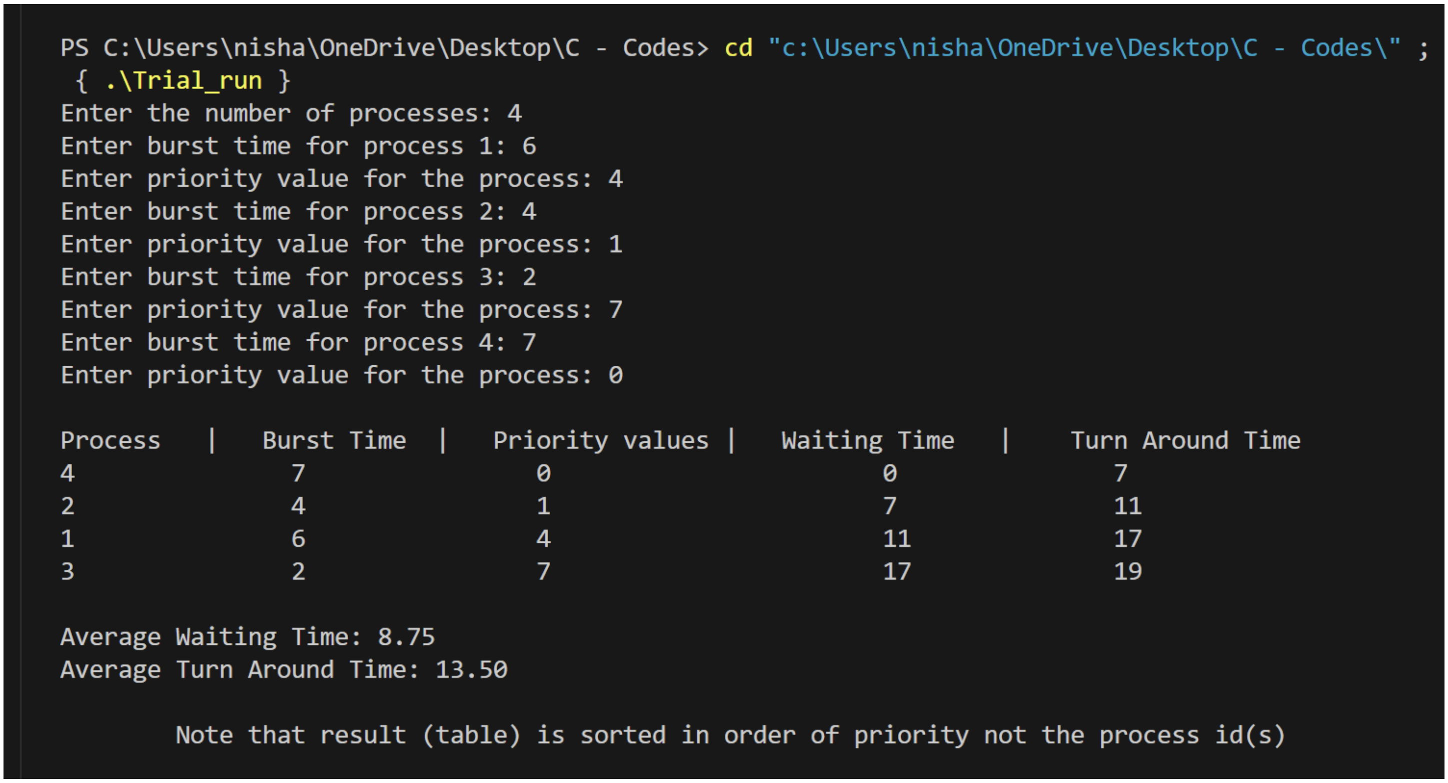
1. Priority Scheduling :

: CODE

```
#include <stdio.h>

struct Process {
    int pid;
    int burstTime;
    int priority;
    int waitingTime;
    int turnAroundTime;
};


void swap(struct Process *a, struct Process *b) {
    struct Process temp = *a;
    *a = *b;
    *b = temp;
}

void sortByPriority(struct Process p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].priority > p[j+1].priority) {
                swap(&p[j], &p[j+1]);
            }
        }
    }
}


void calculateWaitingTime(struct Process p[], int n) {
    p[0].waitingTime = 0; // First process has no waiting time

    for (int i = 1; i < n; i++) {
        p[i].waitingTime = p[i-1].waitingTime + p[i-1].burstTime;
    }
}


void calculateTurnAroundTime(struct Process p[], int n) {
    for (int i = 0; i < n; i++) {
        p[i].turnAroundTime = p[i].waitingTime + p[i].burstTime;
    }
}

void calculateAverageTimes(struct Process p[], int n) {
    float totalWaitingTime = 0, totalTurnAroundTime = 0;

    calculateWaitingTime(p, n);
    calculateTurnAroundTime(p, n);

    printf("\nProcess   |   Burst Time  |   Priority values |   Waiting Time   |    Turn
Around Time\n");
    for (int i = 0; i < n; i++) {
        totalWaitingTime += p[i].waitingTime;
        totalTurnAroundTime += p[i].turnAroundTime;
        printf("%d\t\t%d \t\t %d \t\t\t %d \t\t %d\n", p[i].pid, p[i].burstTime,
p[i].priority, p[i].waitingTime, p[i].turnAroundTime);
    }

    printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
    printf("Average Turn Around Time: %.2f\n", totalTurnAroundTime / n);
}

int main() {
    int n;

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

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].pid = i+1;
        printf("Enter burst time for process %d: ", p[i].pid);
        scanf("%d", &p[i].burstTime);
        printf("Enter priority value for the process: ", p[i].pid);
        scanf("%d",&p[i].priority);
    }


    sortByPriority(p, n);

    calculateAverageTimes(p, n);

    printf("\n\tNote that result (table) is sorted in order of priority not the process
id(s)\n\n.");
    return 0;
}
```

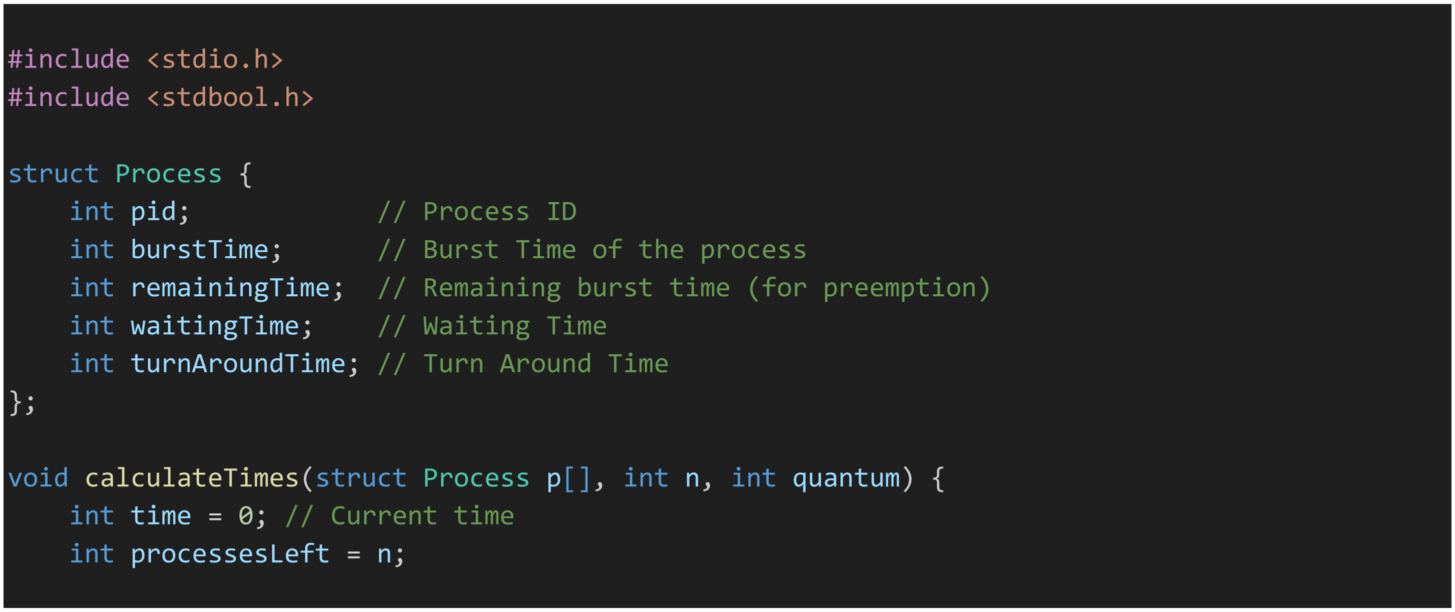
: Result



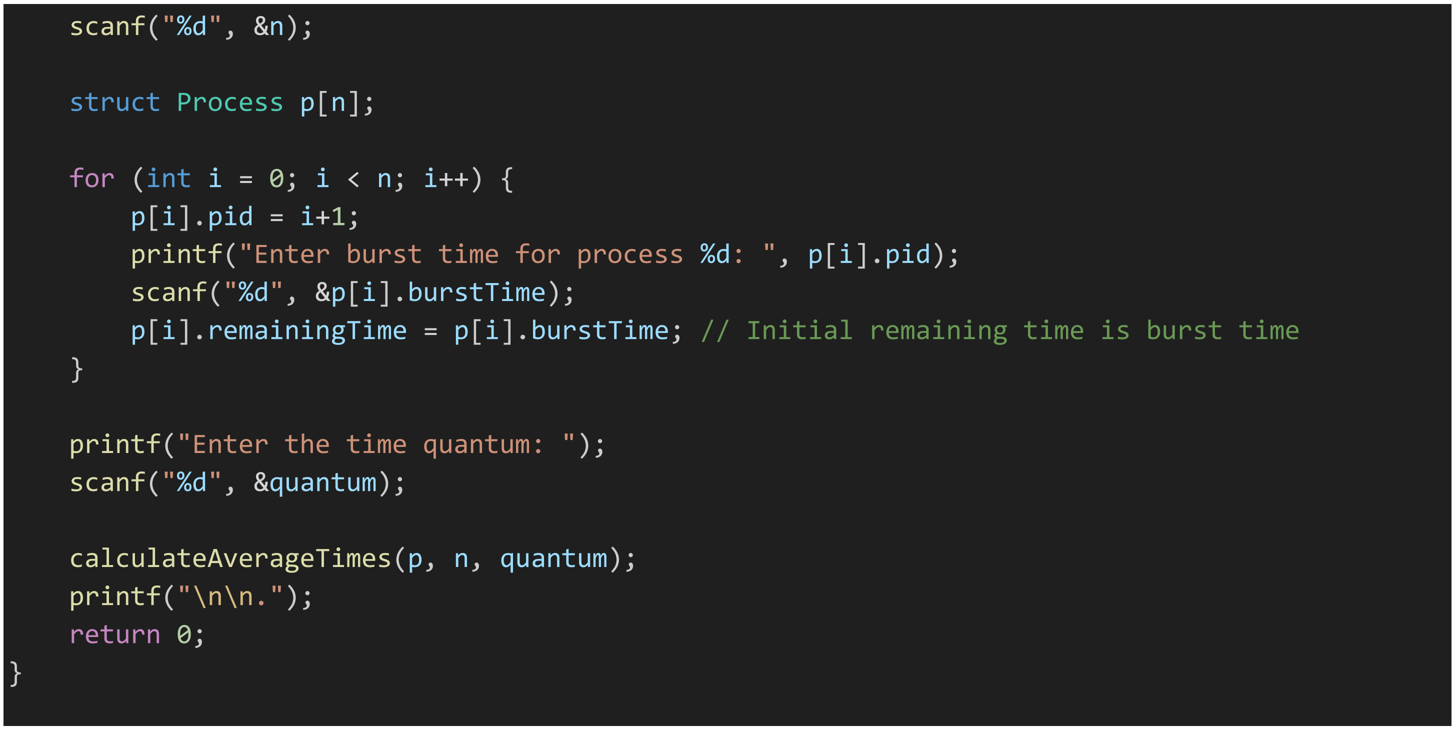
```
PS C:\Users\nisha\OneDrive\Desktop\C - Codes> cd "c:\Users\nisha\OneDrive\Desktop\C - Codes\" ;
 { .\Trial_run }
Enter the number of processes: 4
Enter burst time for process 1: 6
Enter priority value for the process: 4
Enter burst time for process 2: 4
Enter priority value for the process: 1
Enter burst time for process 3: 2
Enter priority value for the process: 7
Enter burst time for process 4: 7
Enter priority value for the process: 0

Process    |   Burst Time  |   Priority values |   Waiting Time    |     Turn Around Time
4              7               0                      0                  7
2              4               1                      7                  11
1              6               4                      11                 17
3              2               7                      17                 19

Average Waiting Time: 8.75
Average Turn Around Time: 13.50

        Note that result (table) is sorted in order of priority not the process id(s)
```

2. Round-Robin Scheduling :

: CODE

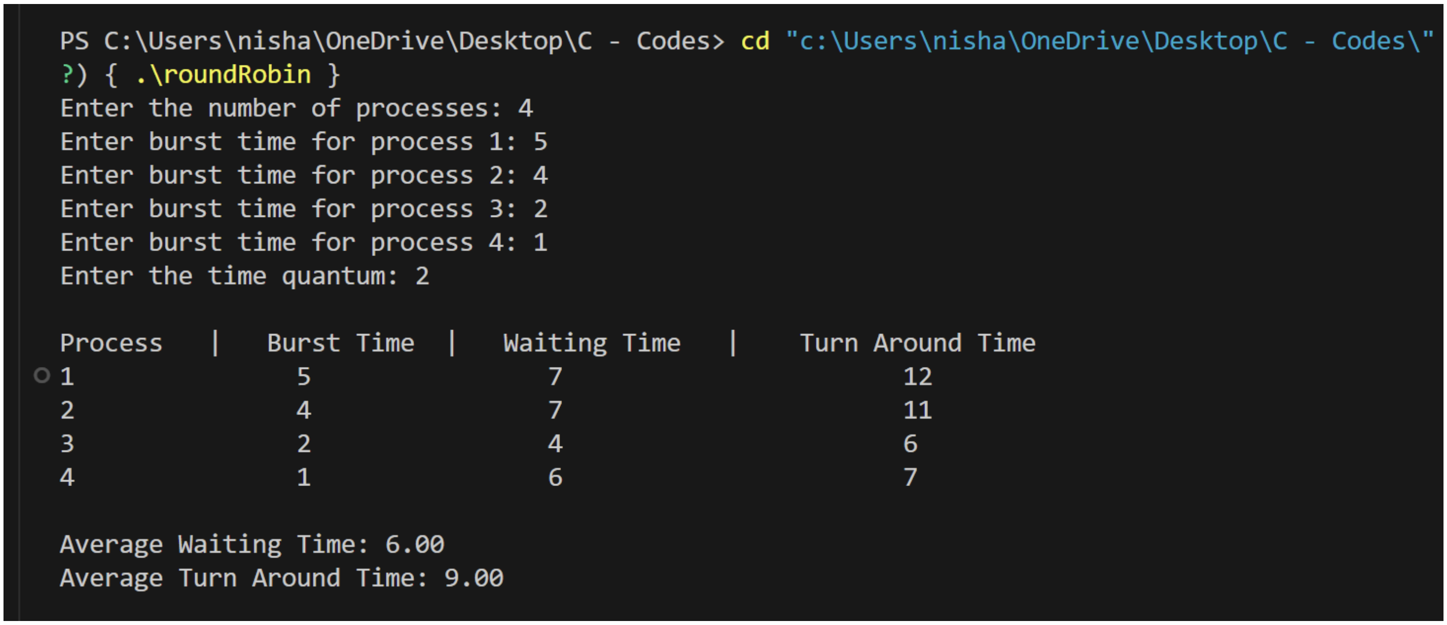


```
#include <stdio.h>
#include <stdbool.h>

struct Process {
    int pid;            // Process ID
    int burstTime;      // Burst Time of the process
    int remainingTime;  // Remaining burst time (for preemption)
    int waitingTime;    // Waiting Time
    int turnAroundTime; // Turn Around Time
};

void calculateTimes(struct Process p[], int n, int quantum) {
    int time = 0; // Current time
    int processesLeft = n;
    while (processesLeft > 0) {
        bool allDone = true;

        for (int i = 0; i < n; i++) {
            if (p[i].remainingTime > 0) {
                allDone = false; // There's still at least one process left
                if (p[i].remainingTime > quantum) { // Process is preempted after 'quantum'
time
                    time += quantum;
                    p[i].remainingTime -= quantum;
                } else {    // Process finishes execution
                    time += p[i].remainingTime;
                    p[i].waitingTime = time - p[i].burstTime;
                    p[i].remainingTime = 0;
                    processesLeft--;
                }
            }
        }

        if (allDone)
            break;
    }

    // Calculate turn around time for each process
    for (int i = 0; i < n; i++) {
        p[i].turnAroundTime = p[i].waitingTime + p[i].burstTime;
    }
}

void calculateAverageTimes(struct Process p[], int n, int quantum) {
    float totalWaitingTime = 0, totalTurnAroundTime = 0;

    calculateTimes(p, n, quantum);

    printf("\nProcess   |   Burst Time  |   Waiting Time   |    Turn Around Time\n");
    for (int i = 0; i < n; i++) {
        totalWaitingTime += p[i].waitingTime;
        totalTurnAroundTime += p[i].turnAroundTime;
        printf("%d\t\t%d \t\t %d \t\t\t %d\n", p[i].pid, p[i].burstTime, p[i].waitingTime,
p[i].turnAroundTime);
    }

    printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
    printf("Average Turn Around Time: %.2f\n", totalTurnAroundTime / n);
}

int main() {
    int n, quantum;

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



```c
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].pid = i+1;
        printf("Enter burst time for process %d: ", p[i].pid);
        scanf("%d", &p[i].burstTime);
        p[i].remainingTime = p[i].burstTime; // Initial remaining time is burst time
    }

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

    calculateAverageTimes(p, n, quantum);
    printf("\n\n.");
    return 0;
}
```

: OUTPUT



```
PS C:\Users\nisha\OneDrive\Desktop\C - Codes> cd "c:\Users\nisha\OneDrive\Desktop\C - Codes\"
?) { .\roundRobin }
Enter the number of processes: 4
Enter burst time for process 1: 5
Enter burst time for process 2: 4
Enter burst time for process 3: 2
Enter burst time for process 4: 1
Enter the time quantum: 2

Process    |   Burst Time  |   Waiting Time    |    Turn Around Time
1              5               7                       12
2              4               7                       11
3              2               4                       6
4              1               6                       7

Average Waiting Time: 6.00
Average Turn Around Time: 9.00
```