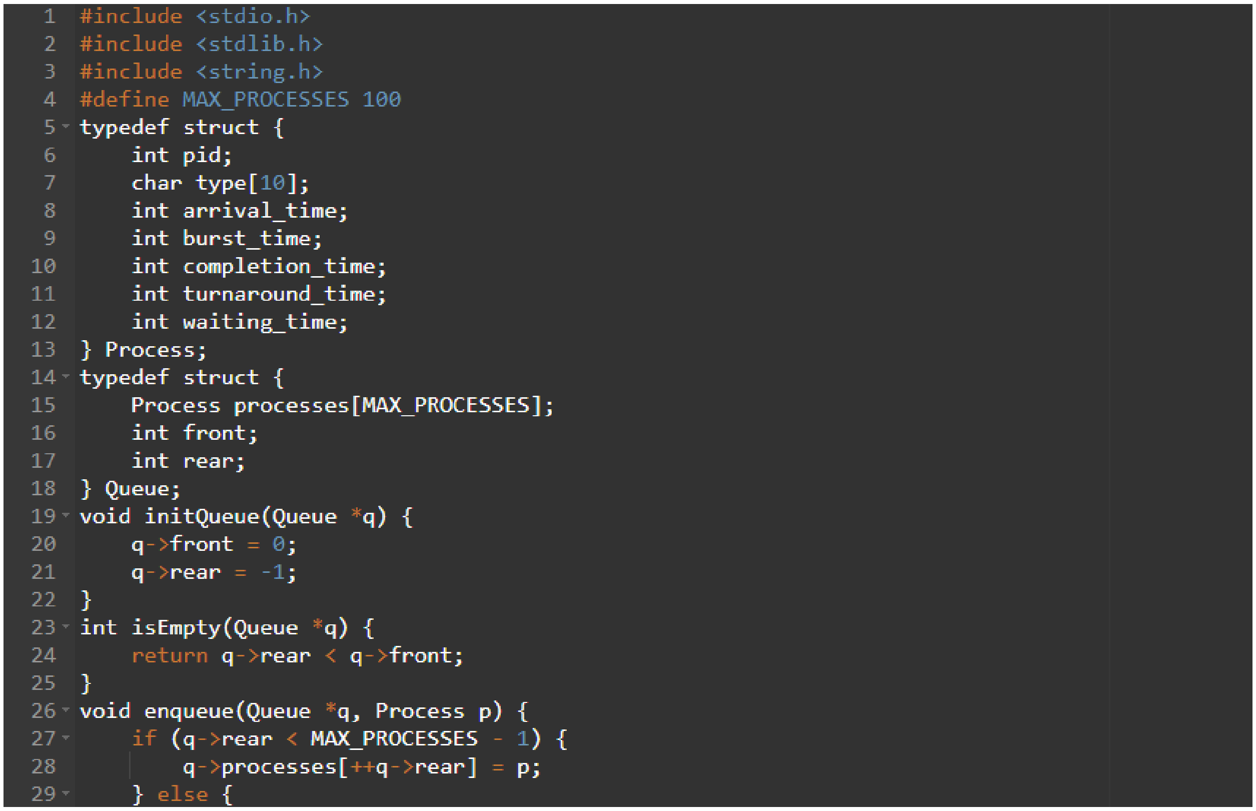
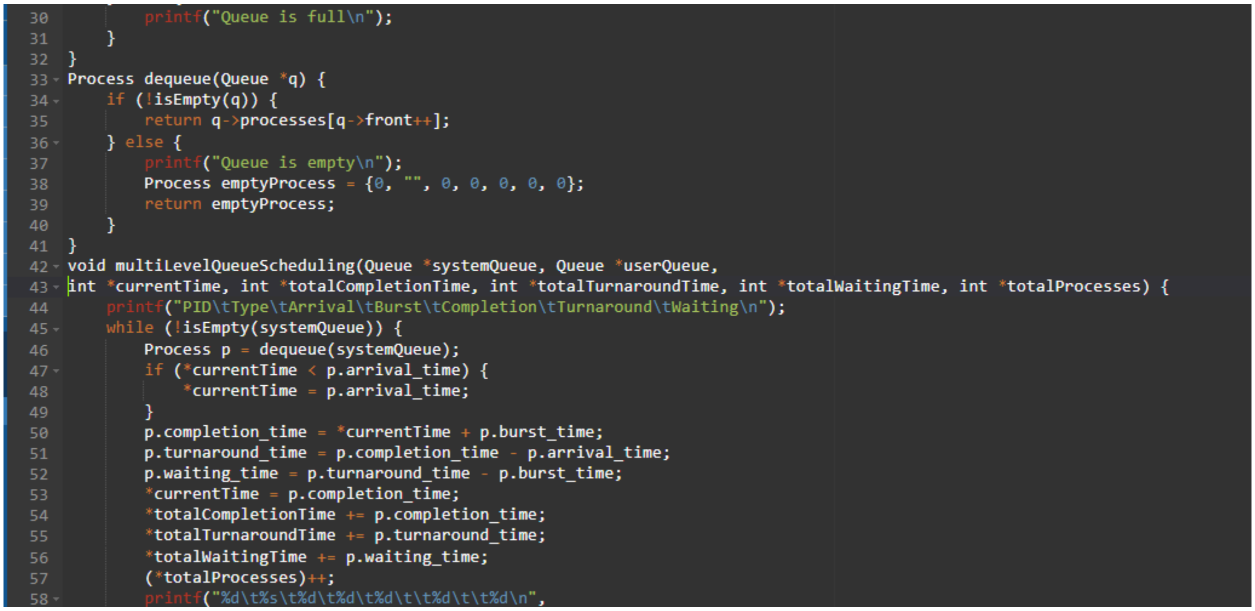
Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

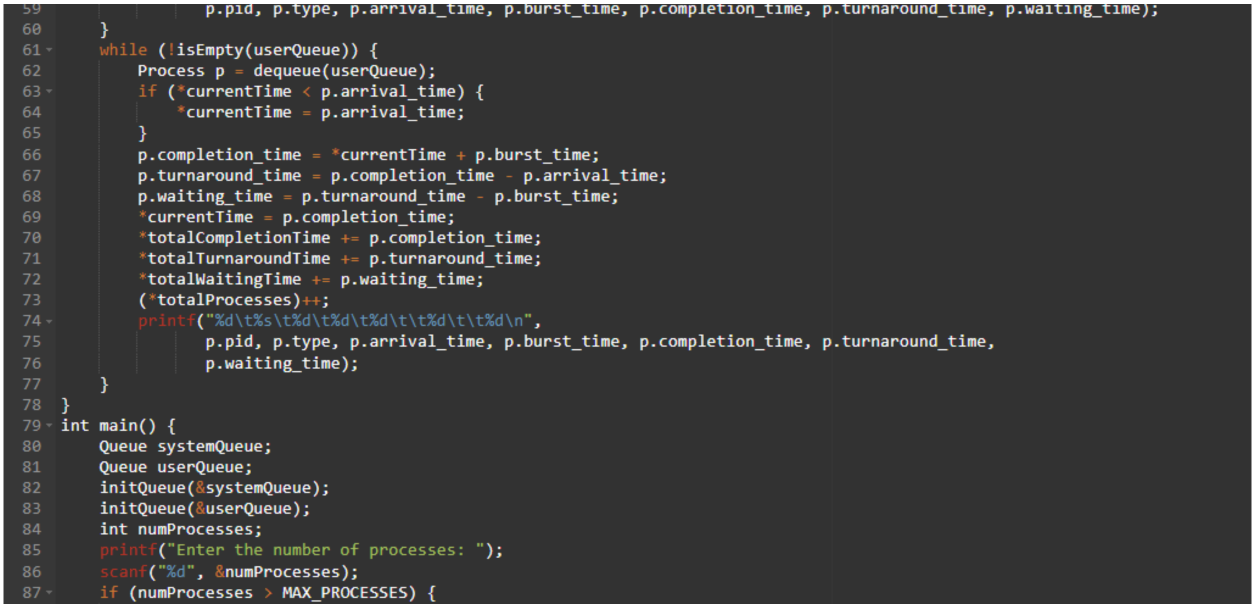
Input



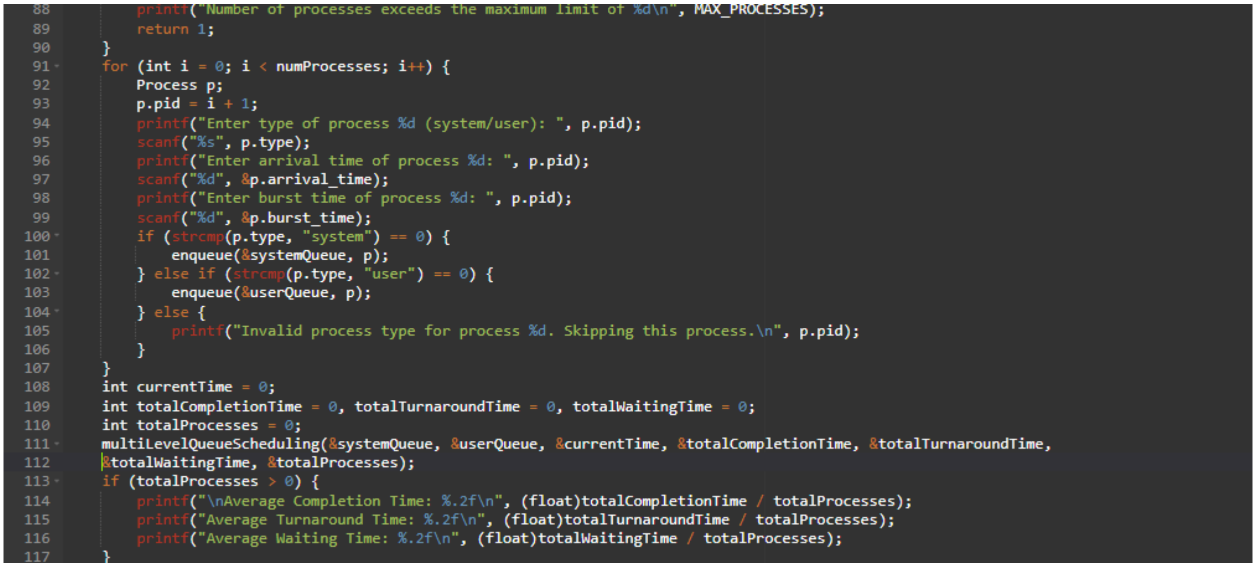
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_PROCESSES 100
typedef struct {
    int pid;
    char type[10];
    int arrival_time;
    int burst_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
} Process;
typedef struct {
    Process processes[MAX_PROCESSES];
    int front;
    int rear;
} Queue;
void initQueue(Queue *q) {
    q->front = 0;
    q->rear = -1;
}
int isEmpty(Queue *q) {
    return q->rear < q->front;
}
void enqueue(Queue *q, Process p) {
    if (q->rear < MAX_PROCESSES - 1) {
        q->processes[++q->rear] = p;
    } else {
```



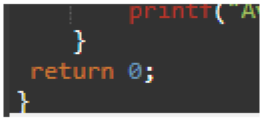
```c
        printf("Queue is full\n");
    }
}
Process dequeue(Queue *q) {
    if (!isEmpty(q)) {
        return q->processes[q->front++];
    } else {
        printf("Queue is empty\n");
        Process emptyProcess = {0, "", 0, 0, 0, 0, 0};
        return emptyProcess;
    }
}
void multiLevelQueueScheduling(Queue *systemQueue, Queue *userQueue,
int *currentTime, int *totalCompletionTime, int *totalTurnaroundTime, int *totalWaitingTime, int *totalProcesses) {
    printf("PID\tType\tArrival\tBurst\tCompletion\tTurnaround\tWaiting\n");
    while (!isEmpty(systemQueue)) {
        Process p = dequeue(systemQueue);
        if (*currentTime < p.arrival_time) {
            *currentTime = p.arrival_time;
        }
        p.completion_time = *currentTime + p.burst_time;
        p.turnaround_time = p.completion_time - p.arrival_time;
        p.waiting_time = p.turnaround_time - p.burst_time;
        *currentTime = p.completion_time;
        *totalCompletionTime += p.completion_time;
        *totalTurnaroundTime += p.turnaround_time;
        *totalWaitingTime += p.waiting_time;
        (*totalProcesses)++;
        printf("%d\t%s\t%d\t%d\t%d\t\t%d\t\t%d\n",
```



```
            p.pid, p.type, p.arrival_time, p.burst_time, p.completion_time, p.turnaround_time, p.waiting_time);
    }
    while (!isEmpty(userQueue)) {
        Process p = dequeue(userQueue);
        if (*currentTime < p.arrival_time) {
            *currentTime = p.arrival_time;
        }
        p.completion_time = *currentTime + p.burst_time;
        p.turnaround_time = p.completion_time - p.arrival_time;
        p.waiting_time = p.turnaround_time - p.burst_time;
        *currentTime = p.completion_time;
        *totalCompletionTime += p.completion_time;
        *totalTurnaroundTime += p.turnaround_time;
        *totalWaitingTime += p.waiting_time;
        (*totalProcesses)++;
        printf("%d\t%s\t%d\t%d\t%d\t\t%d\t\t%d\n",
            p.pid, p.type, p.arrival_time, p.burst_time, p.completion_time, p.turnaround_time,
            p.waiting_time);
    }
}
int main() {
    Queue systemQueue;
    Queue userQueue;
    initQueue(&systemQueue);
    initQueue(&userQueue);
    int numProcesses;
    printf("Enter the number of processes: ");
    scanf("%d", &numProcesses);
    if (numProcesses > MAX_PROCESSES) {
```

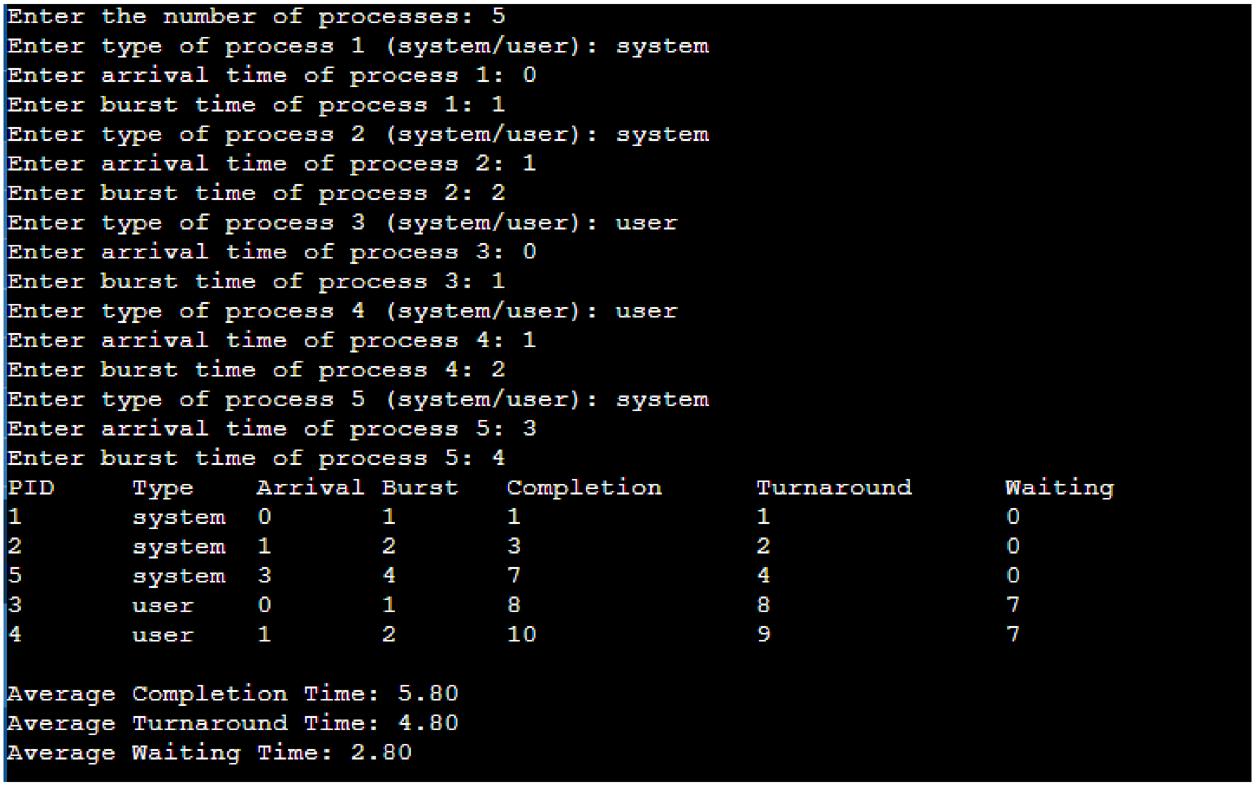


```
        printf("Number of processes exceeds the maximum limit of %d\n", MAX_PROCESSES);
        return 1;
    }
    for (int i = 0; i < numProcesses; i++) {
        Process p;
        p.pid = i + 1;
        printf("Enter type of process %d (system/user): ", p.pid);
        scanf("%s", p.type);
        printf("Enter arrival time of process %d: ", p.pid);
        scanf("%d", &p.arrival_time);
        printf("Enter burst time of process %d: ", p.pid);
        scanf("%d", &p.burst_time);
        if (strcmp(p.type, "system") == 0) {
            enqueue(&systemQueue, p);
        } else if (strcmp(p.type, "user") == 0) {
            enqueue(&userQueue, p);
        } else {
            printf("Invalid process type for process %d. Skipping this process.\n", p.pid);
        }
    }
    int currentTime = 0;
    int totalCompletionTime = 0, totalTurnaroundTime = 0, totalWaitingTime = 0;
    int totalProcesses = 0;
    multiLevelQueueScheduling(&systemQueue, &userQueue, &currentTime, &totalCompletionTime, &totalTurnaroundTime,
    &totalWaitingTime, &totalProcesses);
    if (totalProcesses > 0) {
        printf("\nAverage Completion Time: %.2f\n", (float)totalCompletionTime / totalProcesses);
        printf("Average Turnaround Time: %.2f\n", (float)totalTurnaroundTime / totalProcesses);
        printf("Average Waiting Time: %.2f\n", (float)totalWaitingTime / totalProcesses);
    }
```



```
        printf("A
    }
    return 0;
}
```

output:



```
Enter the number of processes: 5
Enter type of process 1 (system/user): system
Enter arrival time of process 1: 0
Enter burst time of process 1: 1
Enter type of process 2 (system/user): system
Enter arrival time of process 2: 1
Enter burst time of process 2: 2
Enter type of process 3 (system/user): user
Enter arrival time of process 3: 0
Enter burst time of process 3: 1
Enter type of process 4 (system/user): user
Enter arrival time of process 4: 1
Enter burst time of process 4: 2
Enter type of process 5 (system/user): system
Enter arrival time of process 5: 3
Enter burst time of process 5: 4
PID     Type    Arrival Burst   Completion      Turnaround      Waiting
1       system  0       1       1               1               0
2       system  1       2       3               2               0
5       system  3       4       7               4               0
3       user    0       1       8               8               7
4       user    1       2       10              9               7

Average Completion Time: 5.80
Average Turnaround Time: 4.80
Average Waiting Time: 2.80
```