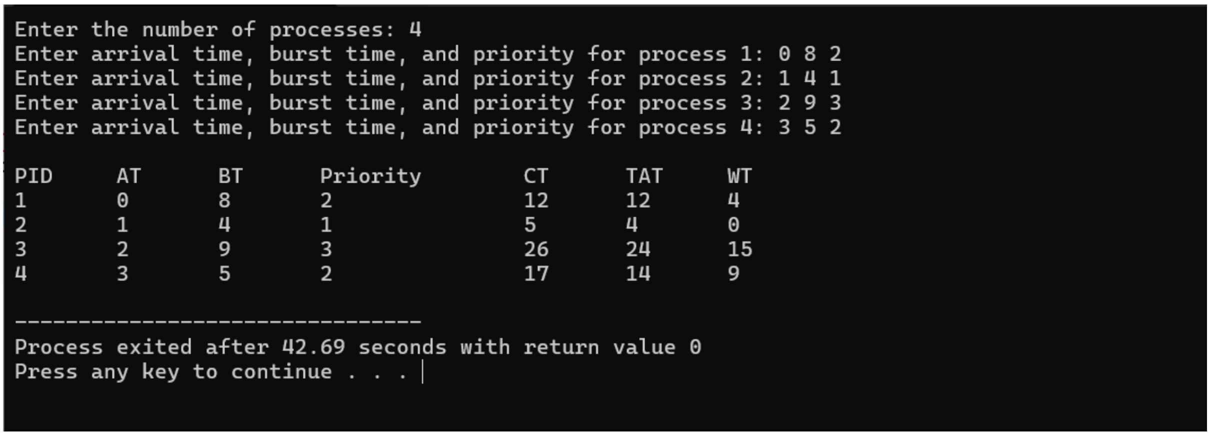
6. Construct a C program to implement preemptive priority scheduling algorithm.

PROGRAM :

```
#include <stdio.h>

#define MAX 100

typedef struct {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int remaining_time;
    int completion_time;
    int waiting_time;
    int turnaround_time;
} Process;

void sortByArrival(Process processes[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].arrival_time > processes[j + 1].arrival_time) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}
void preemptivePriorityScheduling(Process processes[], int n) {
    int completed = 0, time = 0;
    int min_priority_index;
    int is_completed[MAX] = {0};

    while (completed < n) {
        min_priority_index = -1;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time && !is_completed[i]) {
                if (min_priority_index == -1 ||
                    processes[i].priority < processes[min_priority_index].priority ||
                    (processes[i].priority == processes[min_priority_index].priority &&
                     processes[i].arrival_time < processes[min_priority_index].arrival_time)) {
                    min_priority_index = i;
                }
            }
        }
        if (min_priority_index != -1) {
            processes[min_priority_index].remaining_time--;
            time++;
            if (processes[min_priority_index].remaining_time == 0) {
                processes[min_priority_index].completion_time = time;
                processes[min_priority_index].turnaround_time = time -
processes[min_priority_index].arrival_time;
                processes[min_priority_index].waiting_time =
processes[min_priority_index].turnaround_time - processes[min_priority_index].burst_time;
                is_completed[min_priority_index] = 1;
                completed++;
            }
        } else {
            time++;
        }
    }
```

```
}
void displayResults(Process processes[], int n) {
    printf("\nPID\tAT\tBT\tPriority\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t\t%d\t%d\t%d\n",
            processes[i].pid,
            processes[i].arrival_time,
            processes[i].burst_time,
            processes[i].priority,
            processes[i].completion_time,
            processes[i].turnaround_time,
            processes[i].waiting_time);
    }
}
int main() {
    int n;
    Process processes[MAX];
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        processes[i].pid = i + 1;
        printf("Enter arrival time, burst time, and priority for process %d: ", processes[i].pid);
        scanf("%d %d %d", &processes[i].arrival_time, &processes[i].burst_time,
&processes[i].priority);
        processes[i].remaining_time = processes[i].burst_time;
    }
    sortByArrival(processes, n);
    preemptivePriorityScheduling(processes, n);
    displayResults(processes, n);
    return 0;
}
```

OUTPUT:



```
Enter the number of processes: 4
Enter arrival time, burst time, and priority for process 1: 0 8 2
Enter arrival time, burst time, and priority for process 2: 1 4 1
Enter arrival time, burst time, and priority for process 3: 2 9 3
Enter arrival time, burst time, and priority for process 4: 3 5 2

PID     AT      BT      Priority        CT      TAT     WT
1       0       8       2               12      12      4
2       1       4       1               5       4       0
3       2       9       3               26      24      15
4       3       5       2               17      14      9

--------------------------------
Process exited after 42.69 seconds with return value 0
Press any key to continue . . .
```