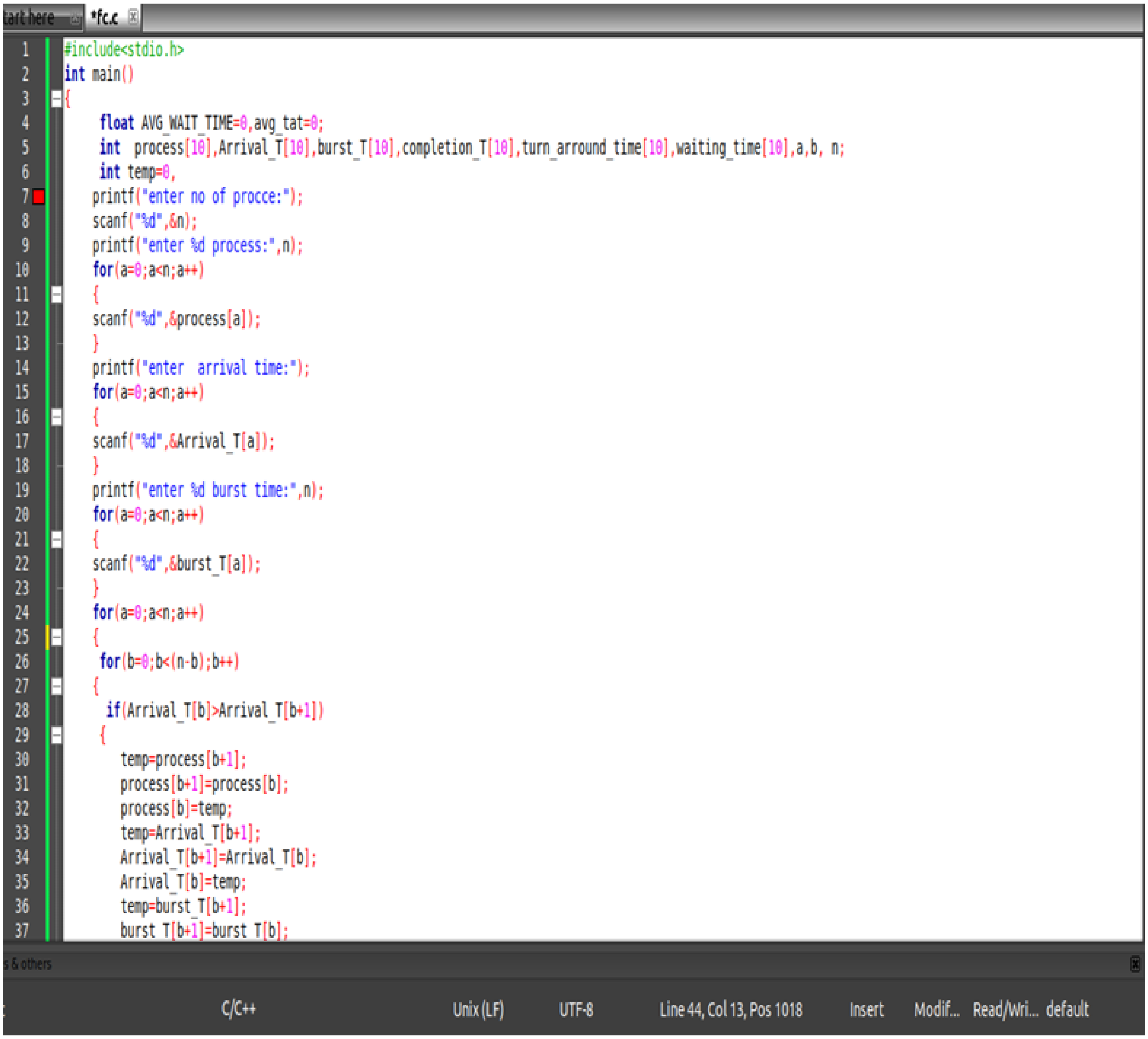
LAB :- 3

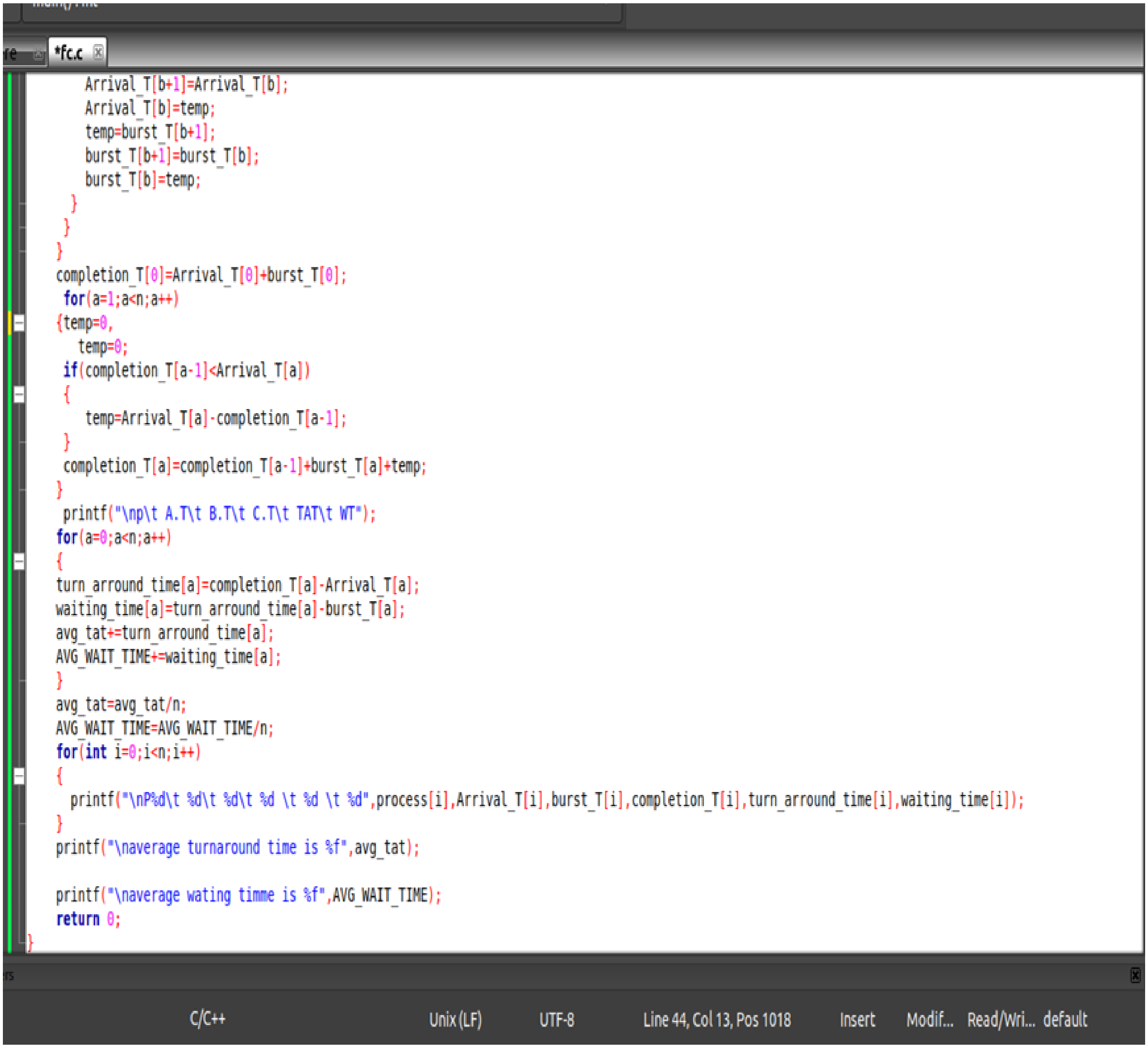
13/10/2022

Create a program that mimics the CPU Scheduling algorithms including multi-level queue scheduling algorithm. Ex: Assume that all 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.

1. FCFS

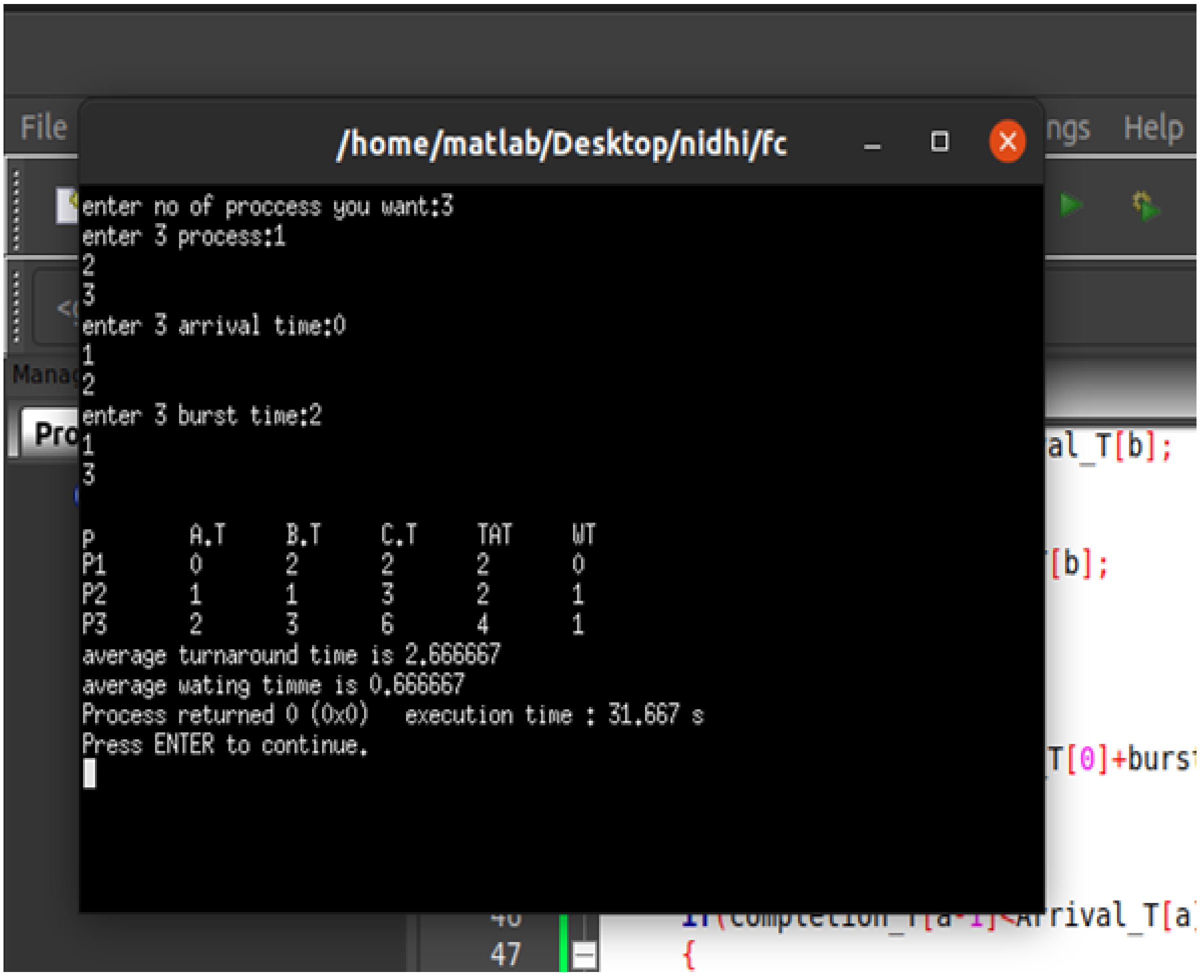


```
#include<stdio.h>
int main()
{
     float AVG_WAIT_TIME=0,avg_tat=0;
     int  process[10],Arrival_T[10],burst_T[10],completion_T[10],turn_arround_time[10],waiting_time[10],a,b, n;
     int temp=0,
    printf("enter no of procce:");
    scanf("%d",&n);
    printf("enter %d process:",n);
    for(a=0;a<n;a++)
    {
    scanf("%d",&process[a]);
    }
    printf("enter  arrival time:");
    for(a=0;a<n;a++)
    {
    scanf("%d",&Arrival_T[a]);
    }
    printf("enter %d burst time:",n);
    for(a=0;a<n;a++)
    {
    scanf("%d",&burst_T[a]);
    }
    for(a=0;a<n;a++)
    {
     for(b=0;b<(n-b);b++)
    {
      if(Arrival_T[b]>Arrival_T[b+1])
     {
        temp=process[b+1];
        process[b+1]=process[b];
        process[b]=temp;
        temp=Arrival_T[b+1];
        Arrival_T[b+1]=Arrival_T[b];
        Arrival_T[b]=temp;
        temp=burst_T[b+1];
        burst_T[b+1]=burst_T[b];
```



```
        Arrival_T[b+1]=Arrival_T[b];
        Arrival_T[b]=temp;
        temp=burst_T[b+1];
        burst_T[b+1]=burst_T[b];
        burst_T[b]=temp;
      }
     }
    }
    completion_T[0]=Arrival_T[0]+burst_T[0];
     for(a=1;a<n;a++)
    {temp=0,
       temp=0;
     if(completion_T[a-1]<Arrival_T[a])
     {
        temp=Arrival_T[a]-completion_T[a-1];
     }
     completion_T[a]=completion_T[a-1]+burst_T[a]+temp;
    }
     printf("\np\t A.T\t B.T\t C.T\t TAT\t WT");
    for(a=0;a<n;a++)
    {
    turn_arround_time[a]=completion_T[a]-Arrival_T[a];
    waiting_time[a]=turn_arround_time[a]-burst_T[a];
    avg_tat+=turn_arround_time[a];
    AVG_WAIT_TIME+=waiting_time[a];
    }
    avg_tat=avg_tat/n;
    AVG_WAIT_TIME=AVG_WAIT_TIME/n;
    for(int i=0;i<n;i++)
    {
      printf("\nP%d\t %d\t %d\t %d \t %d \t %d",process[i],Arrival_T[i],burst_T[i],completion_T[i],turn_arround_time[i],waiting_time[i]);
    }
    printf("\naverage turnaround time is %f",avg_tat);

    printf("\naverage wating timme is %f",AVG_WAIT_TIME);
    return 0;
}
```

Output :-



```
enter no of proccess you want:3
enter 3 process:1
2
3
enter 3 arrival time:0
1
2
enter 3 burst time:2
1
3

P       A.T     B.T     C.T     TAT     WT
P1      0       2       2       2       0
P2      1       1       3       2       1
P3      2       3       6       4       1
average turnaround time is 2.666667
average wating timme is 0.666667
Process returned 0 (0x0)   execution time : 31.667 s
Press ENTER to continue.
```

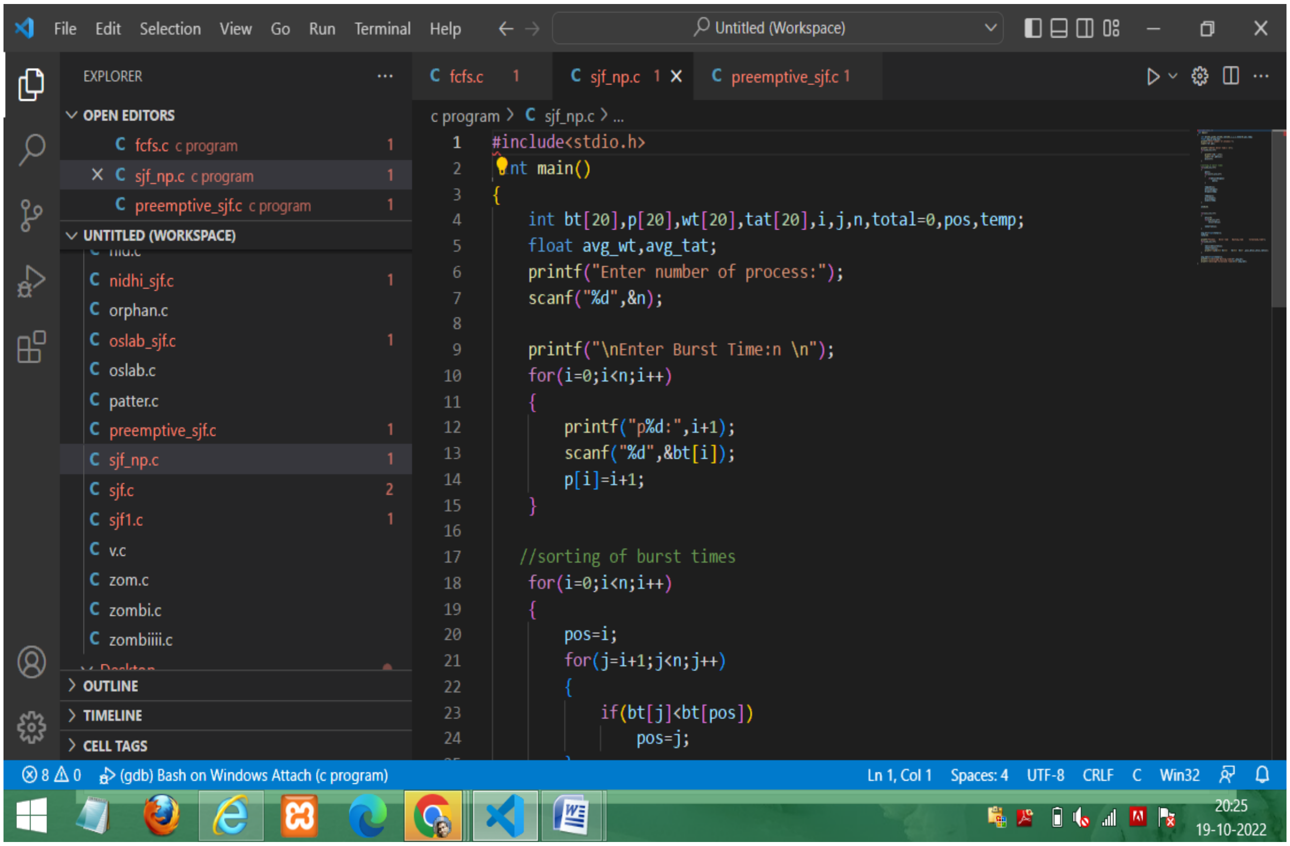
1) Shortest Job First (SJF):

i) Non-Preemptive

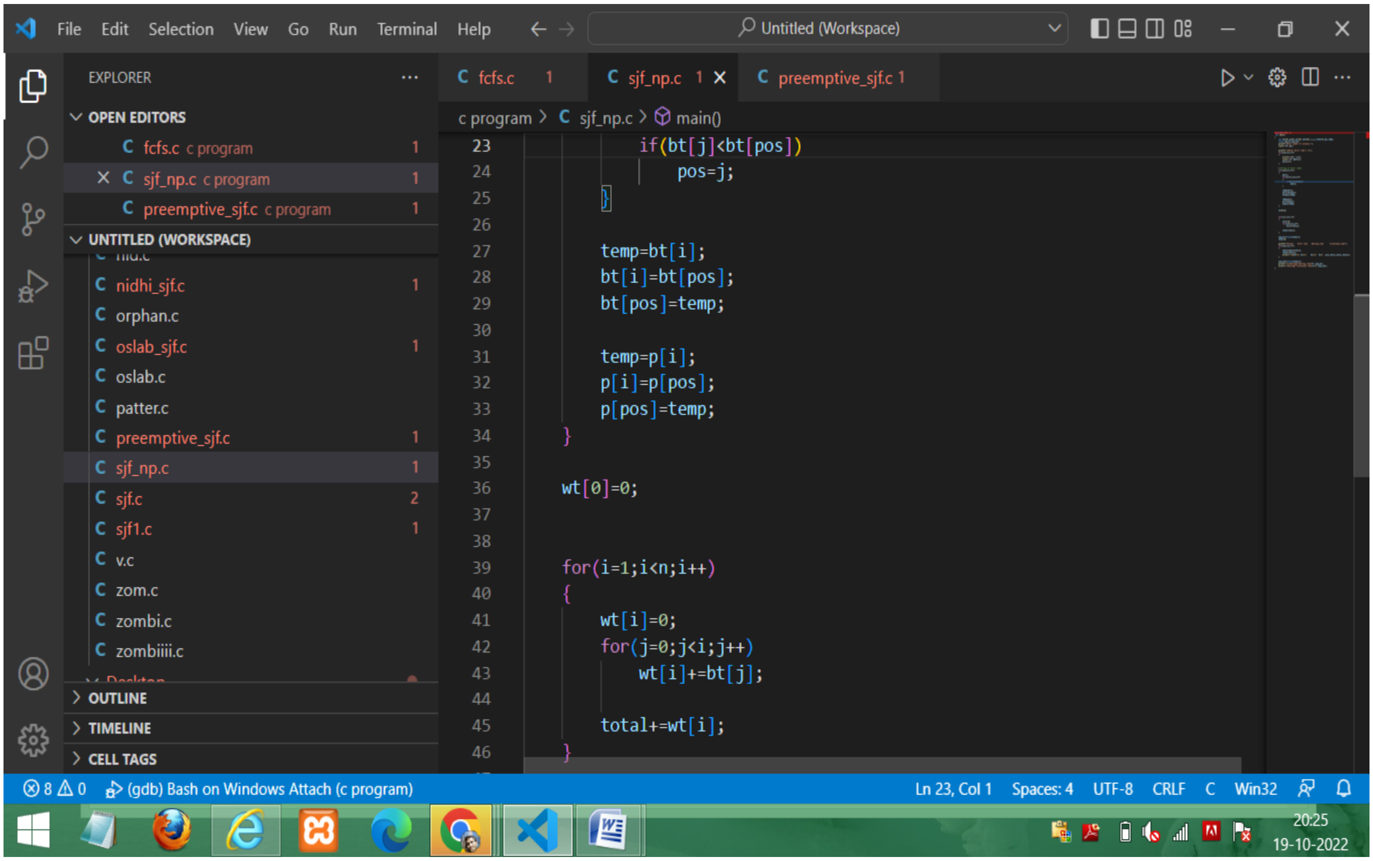
ii) Preemptive

(i). Non-Preemptive SJF Scheduling

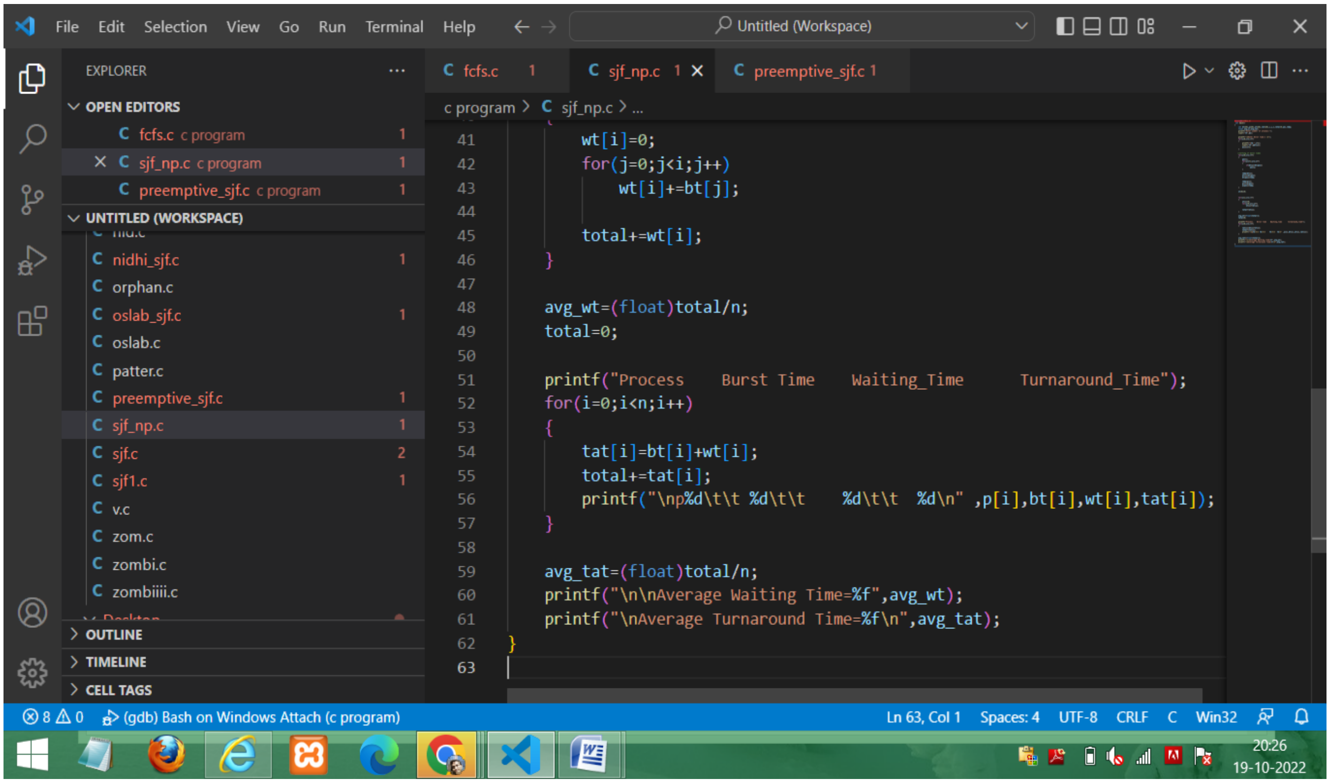
In non-preemptive scheduling, once the CPU cycle is allocated to process, the process holds it till it reaches a waiting state or terminated.Consider the following five processes each having its own unique burst time and arrival time.



```c
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time:n \n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }

   //sorting of burst times
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
```

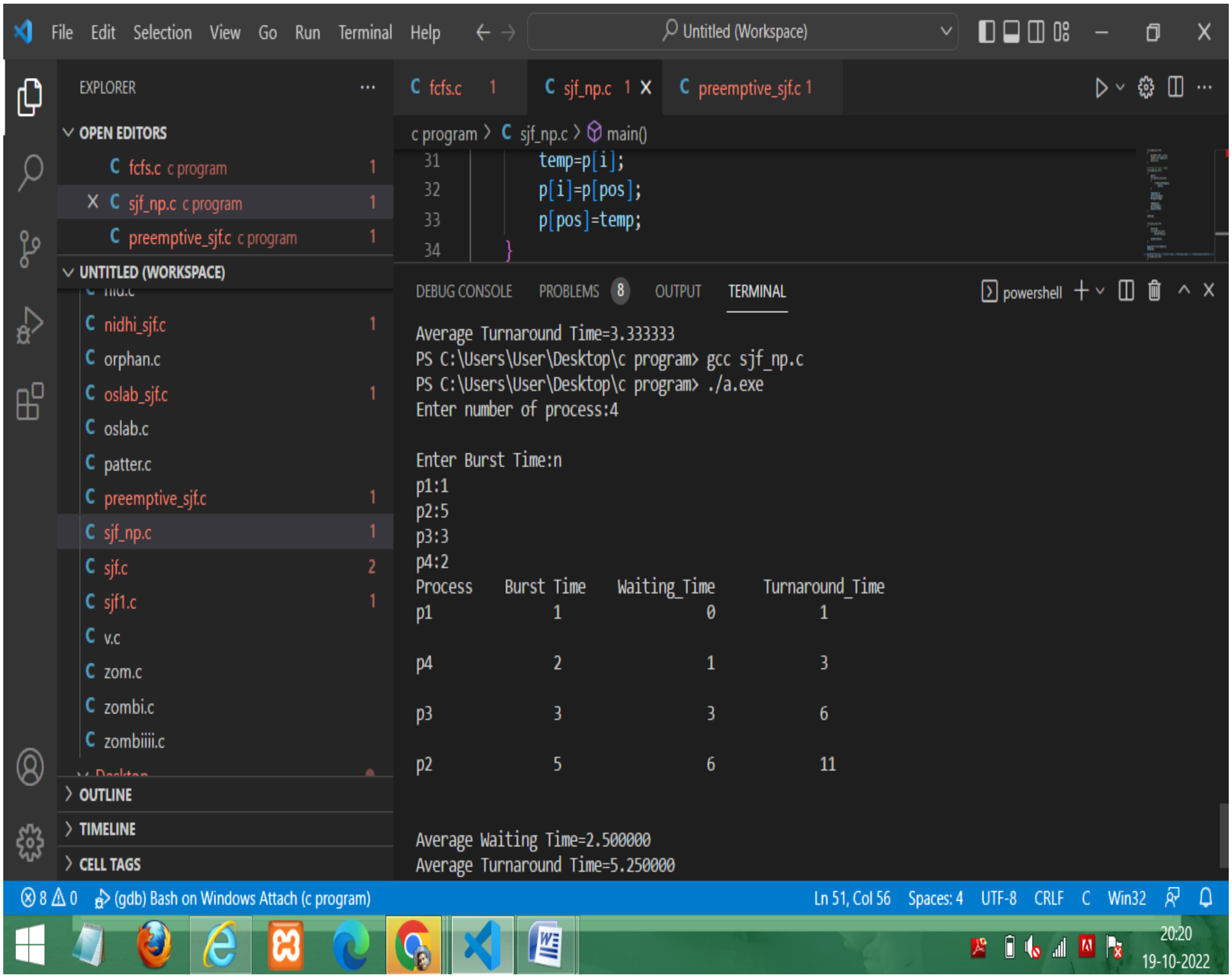


```
            if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;


    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }
```



```
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=(float)total/n;
    total=0;

    printf("Process     Burst Time     Waiting_Time      Turnaround_Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t %d\t\t    %d\t\t  %d\n" ,p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

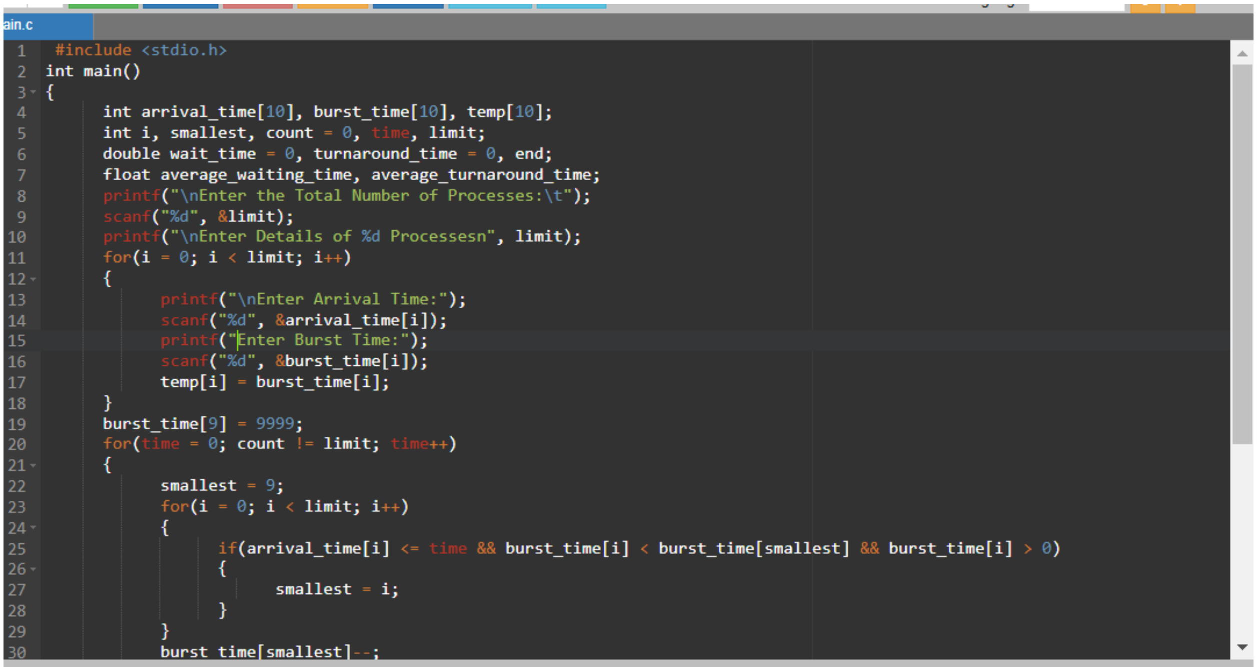
OUTPUT:-



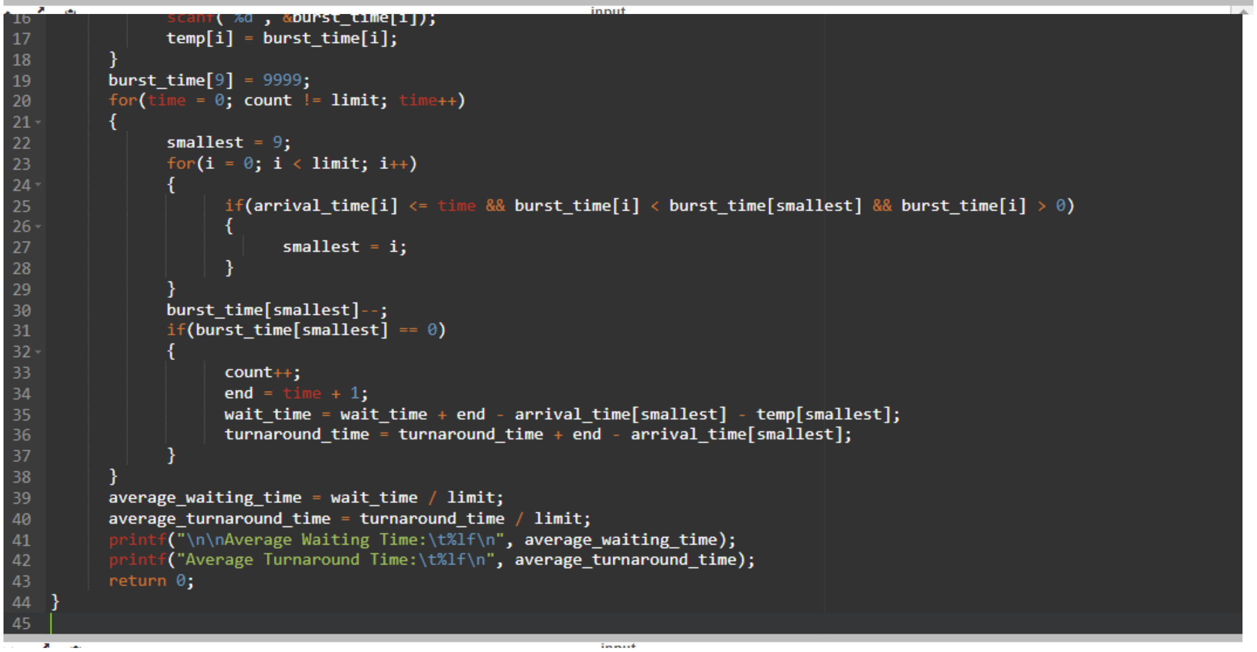
```
c program > sjf_np.c > main()
            temp=p[i];
            p[i]=p[pos];
            p[pos]=temp;
        }
```

```
Average Turnaround Time=3.333333
PS C:\Users\User\Desktop\c program> gcc sjf_np.c
PS C:\Users\User\Desktop\c program> ./a.exe
Enter number of process:4

Enter Burst Time:n
p1:1
p2:5
p3:3
p4:2
Process     Burst Time      Waiting_Time        Turnaround_Time
p1              1               0                   1

p4              2               1                   3

p3              3               3                   6

p2              5               6                   11


Average Waiting Time=2.500000
Average Turnaround Time=5.250000
```

(ii). Preemptive SJF Scheduling



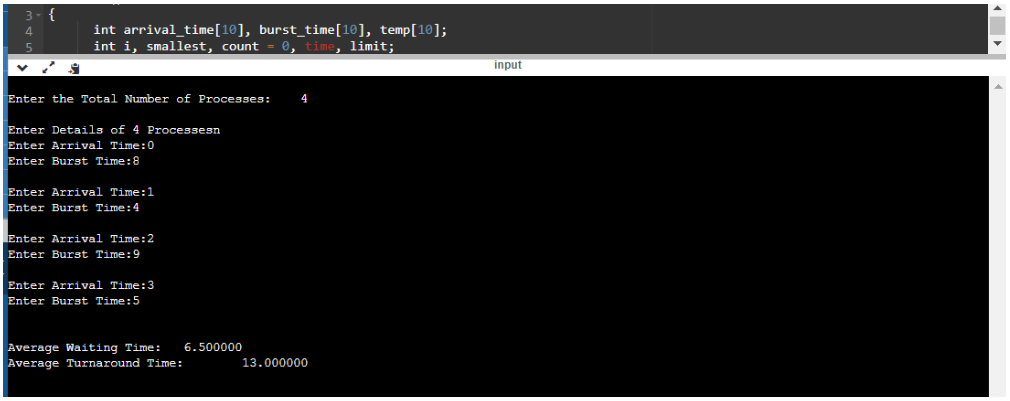
```
 #include <stdio.h>
int main()
{
    int arrival_time[10], burst_time[10], temp[10];
    int i, smallest, count = 0, time, limit;
    double wait_time = 0, turnaround_time = 0, end;
    float average_waiting_time, average_turnaround_time;
    printf("\nEnter the Total Number of Processes:\t");
    scanf("%d", &limit);
    printf("\nEnter Details of %d Processesn", limit);
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Arrival Time:");
        scanf("%d", &arrival_time[i]);
        printf("Enter Burst Time:");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++)
    {
        smallest = 9;
        for(i = 0; i < limit; i++)
        {
            if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] && burst_time[i] > 0)
            {
                smallest = i;
            }
        }
        burst_time[smallest]--;
```



```
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++)
    {
        smallest = 9;
        for(i = 0; i < limit; i++)
        {
            if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] && burst_time[i] > 0)
            {
                smallest = i;
            }
        }
        burst_time[smallest]--;
```

```
        if(burst_time[smallest] == 0)
        {
            count++;
            end = time + 1;
            wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
            turnaround_time = turnaround_time + end - arrival_time[smallest];
        }
    }
    average_waiting_time = wait_time / limit;
    average_turnaround_time = turnaround_time / limit;
    printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
    printf("Average Turnaround Time:\t%lf\n", average_turnaround_time);
    return 0;
}
```

Output :-

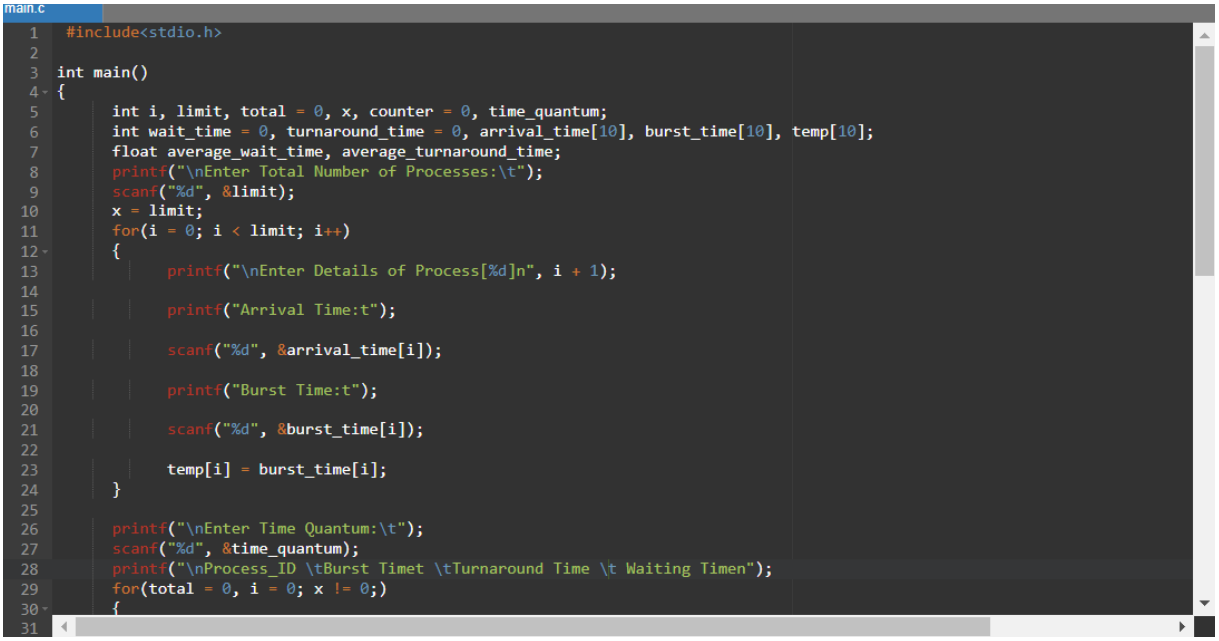


```
{
        int arrival_time[10], burst_time[10], temp[10];
        int i, smallest, count = 0, time, limit;
```

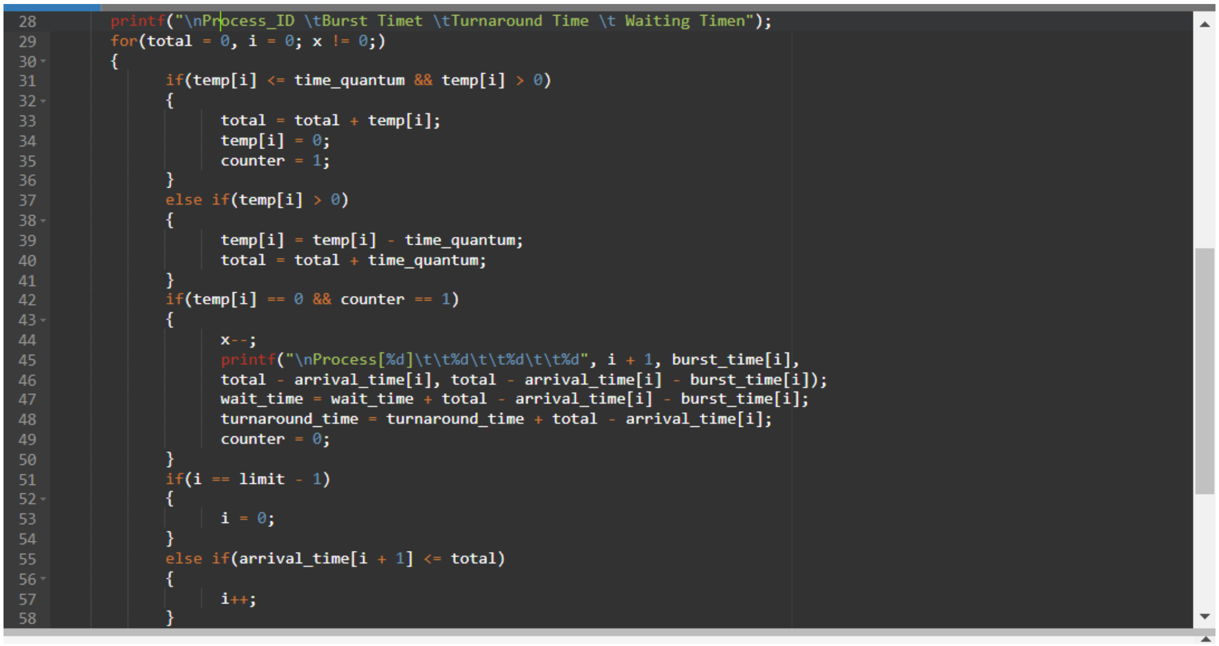
```
Enter the Total Number of Processes:    4

Enter Details of 4 Processesn
Enter Arrival Time:0
Enter Burst Time:8

Enter Arrival Time:1
Enter Burst Time:4

Enter Arrival Time:2
Enter Burst Time:9

Enter Arrival Time:3
Enter Burst Time:5


Average Waiting Time:   6.500000
Average Turnaround Time:        13.000000
```

3. Round Robin scheduling :-

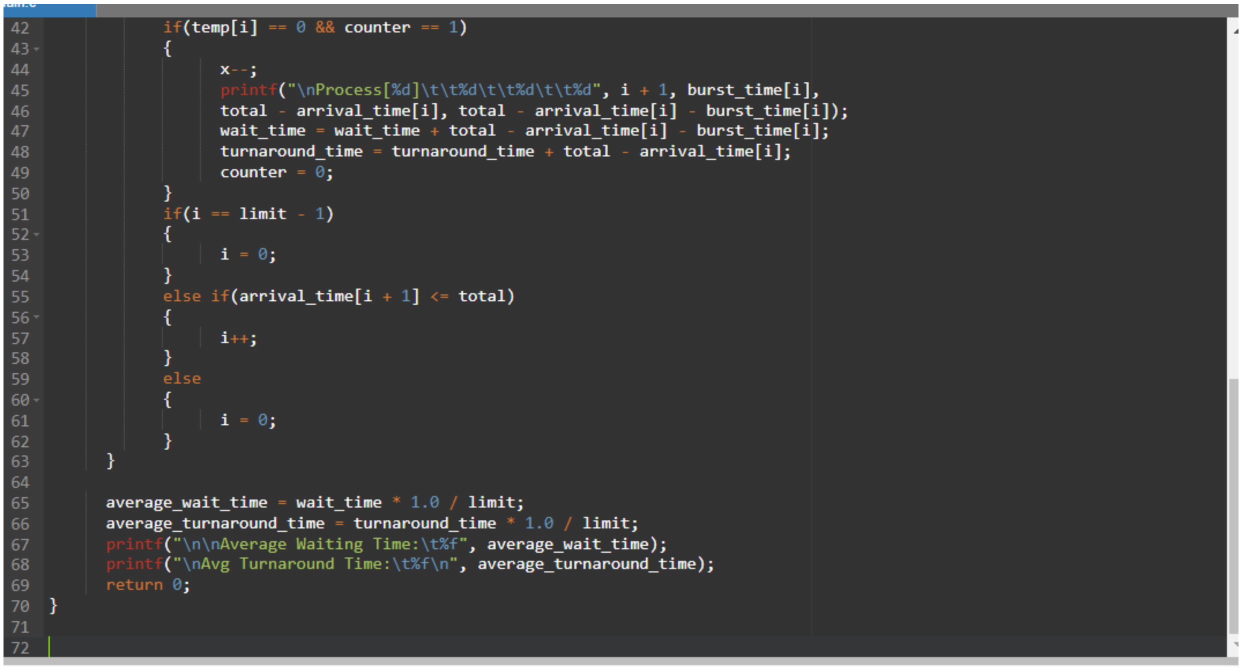
Round Robin Scheduling is a scheduling algorithm used by the system to schedule CPU utilization. This is a preemptive algorithm. There exist a fixed time slice associated with each request called the quantum. The job scheduler saves the progress of the job that is being executed currently and moves to the next job present in the queue when a particular process is executed for a given time quantum.



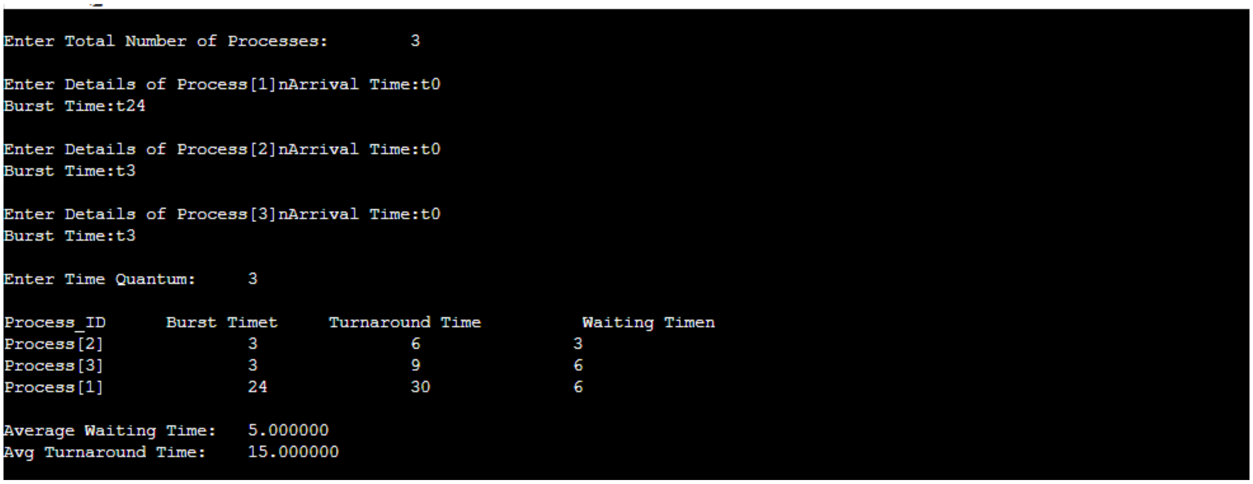
```c
 #include<stdio.h>

int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
    float average_wait_time, average_turnaround_time;
    printf("\nEnter Total Number of Processes:\t");
    scanf("%d", &limit);
    x = limit;
    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Details of Process[%d]n", i + 1);

        printf("Arrival Time:t");

        scanf("%d", &arrival_time[i]);

        printf("Burst Time:t");

        scanf("%d", &burst_time[i]);

        temp[i] = burst_time[i];
    }

    printf("\nEnter Time Quantum:\t");
    scanf("%d", &time_quantum);
    printf("\nProcess_ID \tBurst Timet \tTurnaround Time \t Waiting Timen");
    for(total = 0, i = 0; x != 0;)
    {
```



```c
    printf("\nProcess_ID \tBurst Timet \tTurnaround Time \t Waiting Timen");
    for(total = 0, i = 0; x != 0;)
    {
        if(temp[i] <= time_quantum && temp[i] > 0)
        {
            total = total + temp[i];
            temp[i] = 0;
            counter = 1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - time_quantum;
            total = total + time_quantum;
        }
        if(temp[i] == 0 && counter == 1)
        {
            x--;
            printf("\nProcess[%d]\t\t%d\t\t%d\t\t%d", i + 1, burst_time[i],
            total - arrival_time[i], total - arrival_time[i] - burst_time[i]);
            wait_time = wait_time + total - arrival_time[i] - burst_time[i];
            turnaround_time = turnaround_time + total - arrival_time[i];
            counter = 0;
        }
        if(i == limit - 1)
        {
            i = 0;
        }
        else if(arrival_time[i + 1] <= total)
        {
            i++;
        }
```



```c
            if(temp[i] == 0 && counter == 1)
            {
                x--;
                printf("\nProcess[%d]\t\t%d\t\t%d\t\t%d", i + 1, burst_time[i],
                total - arrival_time[i], total - arrival_time[i] - burst_time[i]);
                wait_time = wait_time + total - arrival_time[i] - burst_time[i];
                turnaround_time = turnaround_time + total - arrival_time[i];
                counter = 0;
            }
            if(i == limit - 1)
            {
                i = 0;
            }
            else if(arrival_time[i + 1] <= total)
            {
                i++;
            }
            else
            {
                i = 0;
            }
    }

    average_wait_time = wait_time * 1.0 / limit;
    average_turnaround_time = turnaround_time * 1.0 / limit;
    printf("\n\nAverage Waiting Time:\t%f", average_wait_time);
    printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);
    return 0;
}

```



```
Enter Total Number of Processes:        3

Enter Details of Process[1]nArrival Time:t0
Burst Time:t24

Enter Details of Process[2]nArrival Time:t0
Burst Time:t3

Enter Details of Process[3]nArrival Time:t0
Burst Time:t3

Enter Time Quantum:     3

Process_ID      Burst Timet     Turnaround Time          Waiting Timen
Process[2]              3               6               3
Process[3]              3               9               6
Process[1]              24              30              6

Average Waiting Time:   5.000000
Avg Turnaround Time:    15.000000
```

4. Priority scheduling

The priority scheduling algorithm follows a method by which a priority is set to the processes available for execution, and the process is selected based on the descending order of priority into the ready queue for execution by the CPU. Several factors can be used to determine the priority value of a process. The factors include the time taken to complete execution, memory spaces required by the process, etc.

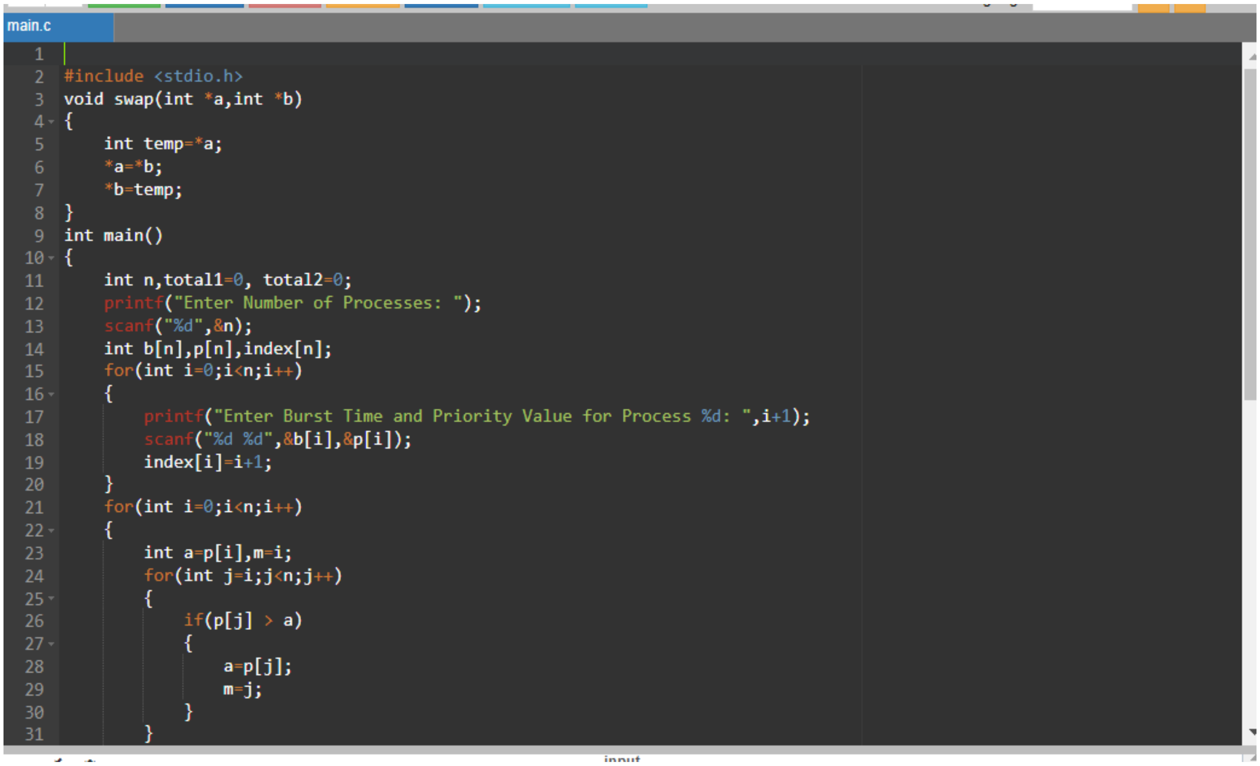
(i). Non preemptive priority scheduling

In the **Preemptive** algorithm, during the execution of a process with high priority, if there is an arrival of another process with a priority higher than the process under execution, then the process currently under execution is stopped, and the new process is allowed to execute.

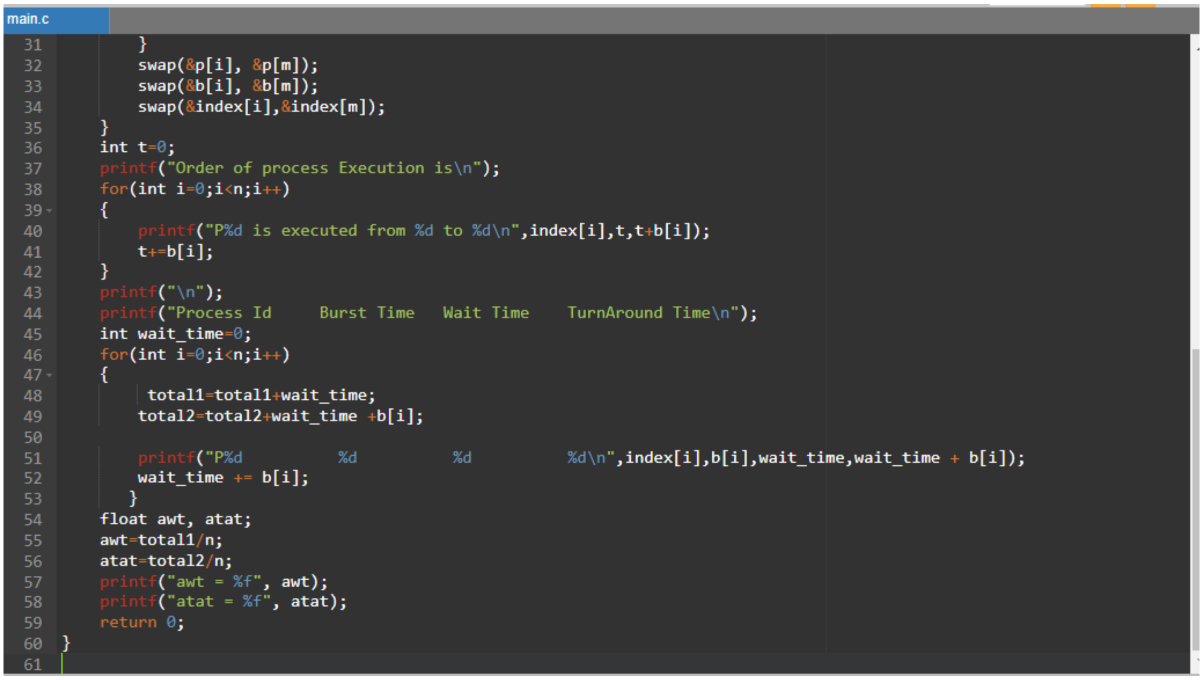
(ii). Preemptive priority scheduling

In the **Non-preemptive** algorithm, the process with the highest priority is allowed to execute in the CPU, and if there is an arrival of another process with a priority higher than the process under execution, then the new process will have to wait until the execution of the current process is completed.

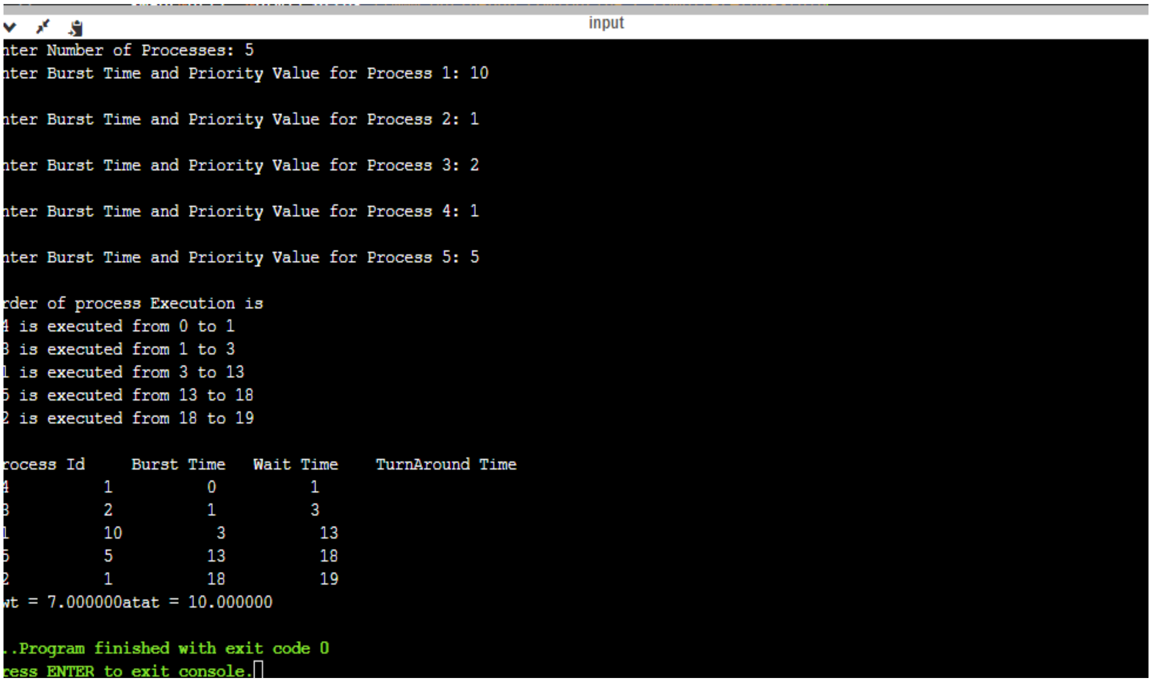
(i). Non preemptive priority scheduling



```

#include <stdio.h>
void swap(int *a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
int main()
{
    int n,total1=0, total2=0;
    printf("Enter Number of Processes: ");
    scanf("%d",&n);
    int b[n],p[n],index[n];
    for(int i=0;i<n;i++)
    {
        printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
        scanf("%d %d",&b[i],&p[i]);
        index[i]=i+1;
    }
    for(int i=0;i<n;i++)
    {
        int a=p[i],m=i;
        for(int j=i;j<n;j++)
        {
            if(p[j] > a)
            {
                a=p[j];
                m=j;
            }
        }
```

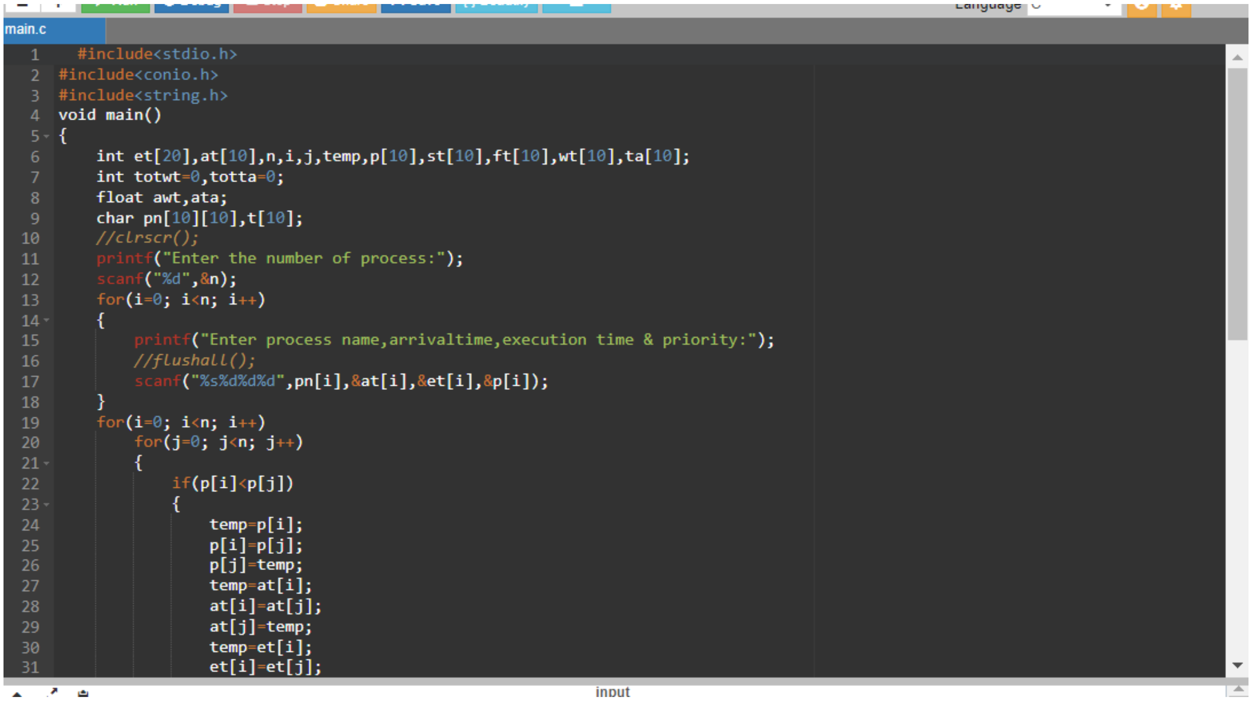


```
        }
        swap(&p[i], &p[m]);
        swap(&b[i], &b[m]);
        swap(&index[i],&index[m]);
    }
    int t=0;
    printf("Order of process Execution is\n");
    for(int i=0;i<n;i++)
    {
        printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
        t+=b[i];
    }
    printf("\n");
    printf("Process Id     Burst Time   Wait Time    TurnAround Time\n");
    int wait_time=0;
    for(int i=0;i<n;i++)
    {
         total1=total1+wait_time;
        total2=total2+wait_time +b[i];

        printf("P%d          %d          %d          %d\n",index[i],b[i],wait_time,wait_time + b[i]);
        wait_time += b[i];
       }
    float awt, atat;
    awt=total1/n;
    atat=total2/n;
    printf("awt = %f", awt);
    printf("atat = %f", atat);
    return 0;
}
```

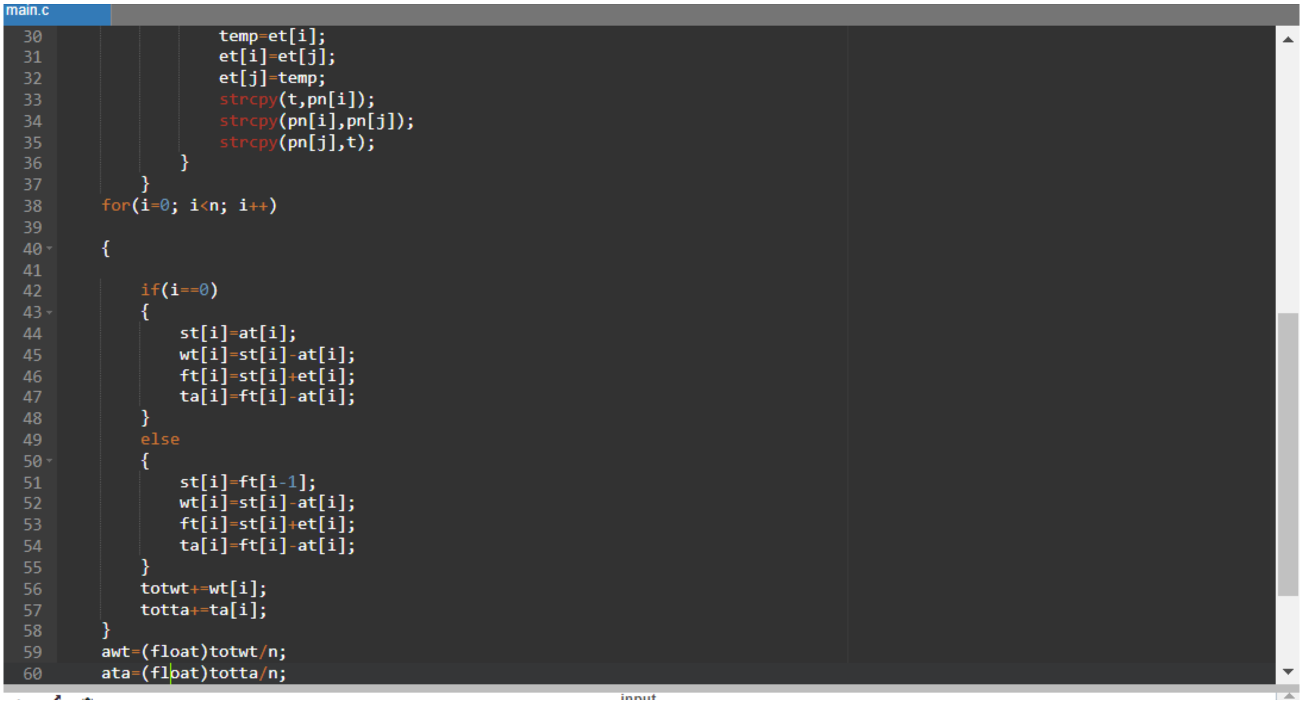


```
nter Number of Processes: 5
nter Burst Time and Priority Value for Process 1: 10

nter Burst Time and Priority Value for Process 2: 1

nter Burst Time and Priority Value for Process 3: 2

nter Burst Time and Priority Value for Process 4: 1

nter Burst Time and Priority Value for Process 5: 5

rder of process Execution is
4 is executed from 0 to 1
3 is executed from 1 to 3
1 is executed from 3 to 13
5 is executed from 13 to 18
2 is executed from 18 to 19

rocess Id     Burst Time   Wait Time    TurnAround Time
4          1          0          1
3          2          1          3
1          10          3          13
5          5          13          18
2          1          18          19
wt = 7.000000atat = 10.000000

..Program finished with exit code 0
ress ENTER to exit console.
```

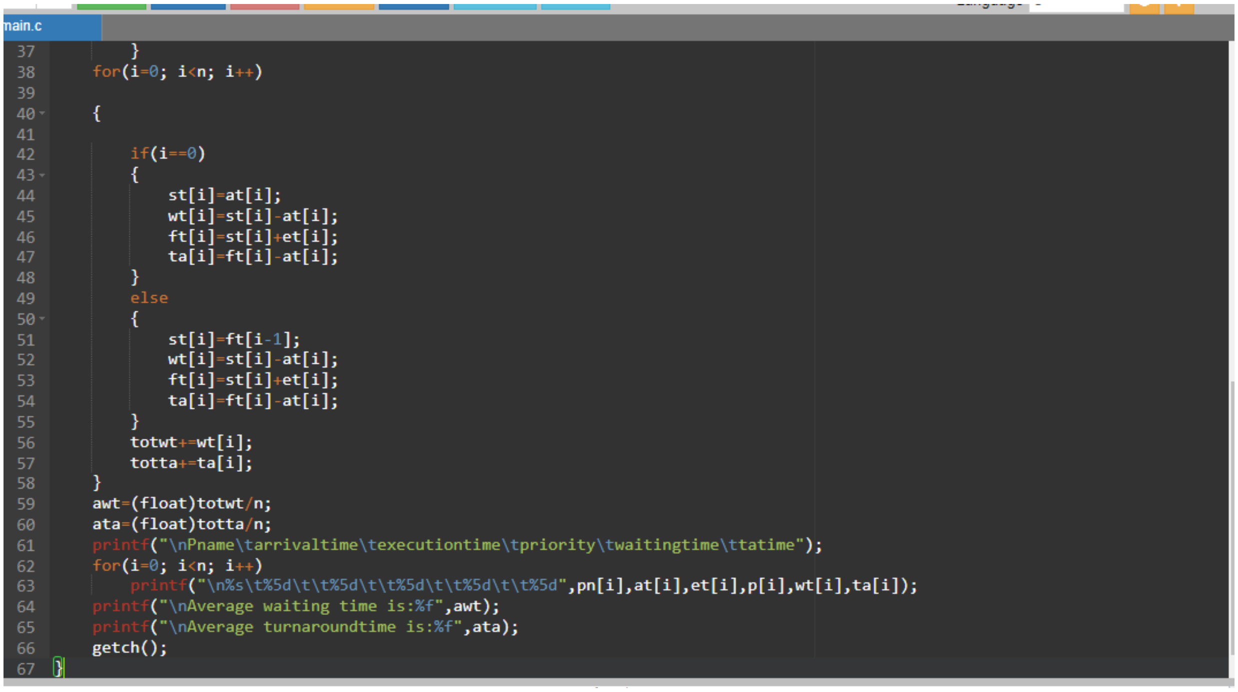
(ii). Pre emptive priority scheduling



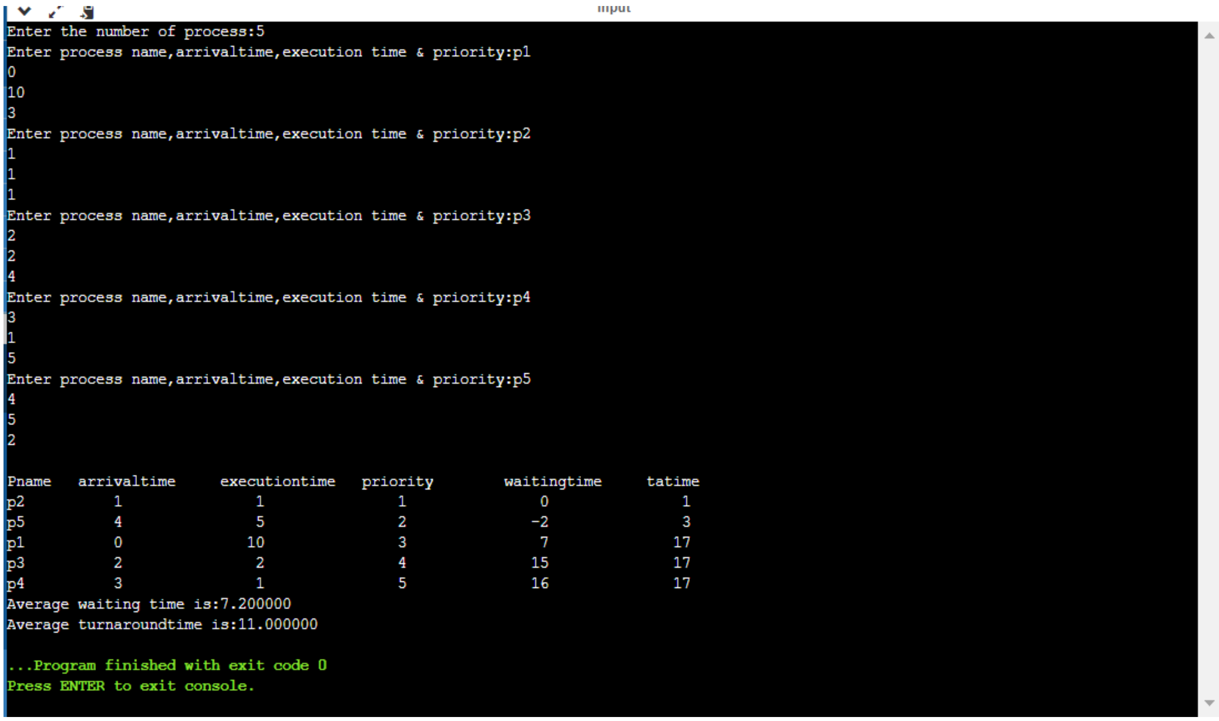
```c
  #include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10];
    int totwt=0,totta=0;
    float awt,ata;
    char pn[10][10],t[10];
    //clrscr();
    printf("Enter the number of process:");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("Enter process name,arrivaltime,execution time & priority:");
        //flushall();
        scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
    }
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
        {
            if(p[i]<p[j])
            {
                temp=p[i];
                p[i]=p[j];
                p[j]=temp;
                temp=at[i];
                at[i]=at[j];
                at[j]=temp;
                temp=et[i];
                et[i]=et[j];
```



```c
                temp=et[i];
                et[i]=et[j];
                et[j]=temp;
                strcpy(t,pn[i]);
                strcpy(pn[i],pn[j]);
                strcpy(pn[j],t);
            }
        }
    for(i=0; i<n; i++)

    {

        if(i==0)
        {
            st[i]=at[i];
            wt[i]=st[i]-at[i];
            ft[i]=st[i]+et[i];
            ta[i]=ft[i]-at[i];
        }
        else
        {
            st[i]=ft[i-1];
            wt[i]=st[i]-at[i];
            ft[i]=st[i]+et[i];
            ta[i]=ft[i]-at[i];
        }
        totwt+=wt[i];
        totta+=ta[i];
    }
    awt=(float)totwt/n;
    ata=(float)totta/n;
```



```
    }
for(i=0; i<n; i++)

{

    if(i==0)
    {
        st[i]=at[i];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+et[i];
        ta[i]=ft[i]-at[i];
    }
    else
    {
        st[i]=ft[i-1];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+et[i];
        ta[i]=ft[i]-at[i];
    }
    totwt+=wt[i];
    totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("\nPname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime");
for(i=0; i<n; i++)
    printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
printf("\nAverage waiting time is:%f",awt);
printf("\nAverage turnaroundtime is:%f",ata);
getch();
}
```



```
Enter the number of process:5
Enter process name,arrivaltime,execution time & priority:p1
0
10
3
Enter process name,arrivaltime,execution time & priority:p2
1
1
1
Enter process name,arrivaltime,execution time & priority:p3
2
2
4
Enter process name,arrivaltime,execution time & priority:p4
3
1
5
Enter process name,arrivaltime,execution time & priority:p5
4
5
2

Pname   arrivaltime     executiontime   priority        waitingtime     tatime
p2          1               1               1               0               1
p5          4               5               2              -2               3
p1          0              10               3               7              17
p3          2               2               4              15              17
p4          3               1               5              16              17
Average waiting time is:7.200000
Average turnaroundtime is:11.000000

...Program finished with exit code 0
Press ENTER to exit console.
```