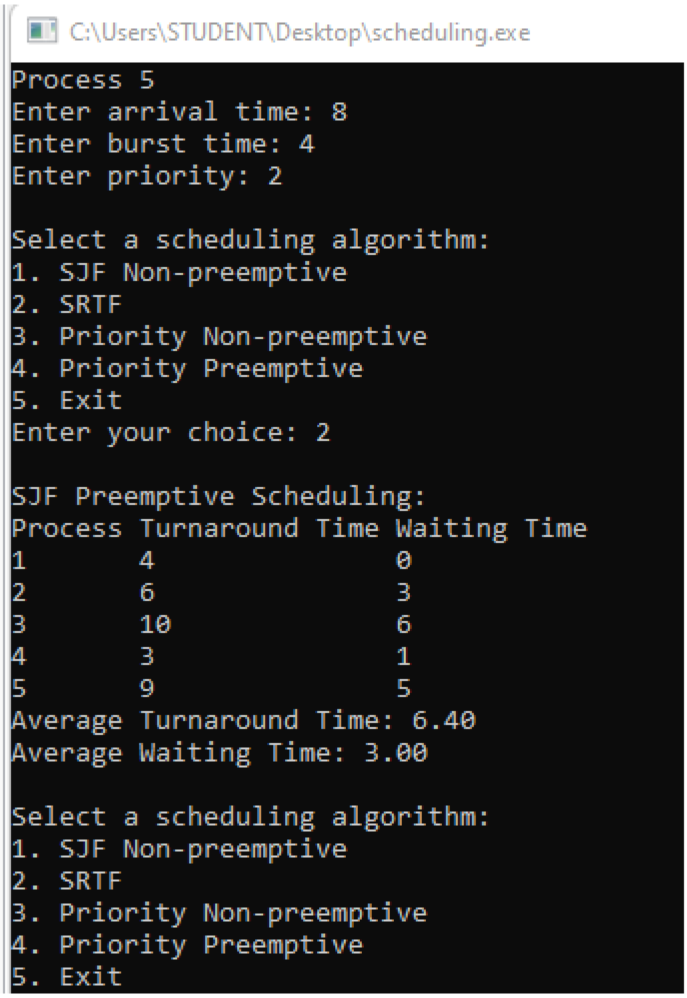
LAB-02

```
#include <stdio.h>

#include <stdbool.h>


#define MAX_PROCESSES 10


struct Process {

    int pid;

    int arrival_time;

    int burst_time;

    int priority;

    int remaining_time;

    int turnaround_time;

    int waiting_time;

};


void sjf_nonpreemptive(struct Process processes[], int n) {

    int i,j,count=0,m;

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

    {

    if(processes[i].arrival_time==0)

    count++;

}

if(count==n||count==1)

{

if(count==n)

{

for (i = 0; i < n - 1; i++) {

        for (j = 0; j < n - i - 1; j++) {

            if (processes[j].burst_time > processes[j + 1].burst_time) {
                struct Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}
else
{
for (i = 1; i < n - 1; i++) {
        for (j = 1; j <= n - i - 1; j++) {
            if (processes[j].burst_time > processes[j + 1].burst_time) {
                struct Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }
}



}

    int total_time = 0;
    double total_turnaround_time = 0;
    double total_waiting_time = 0;

    for (i = 0; i < n; i++) {
        total_time += processes[i].burst_time;
        processes[i].turnaround_time = total_time - processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;
    }

    printf("Process\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time, processes[i].waiting_time);
    }

    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
}

void sjf_preemptive(struct Process processes[], int n) {
    int total_time = 0,i;
    int completed = 0;

    while (completed < n) {
        int shortest_burst = -1;
        int next_process = -1;

        for (i = 0; i < n; i++) {
            if (processes[i].arrival_time <= total_time && processes[i].remaining_time > 0) {
                if (shortest_burst == -1 || processes[i].remaining_time < shortest_burst) {
                    shortest_burst = processes[i].remaining_time;
                    next_process = i;
                }
            }
        }
        if (next_process == -1) {
            total_time++;
            continue;
        }

        processes[next_process].remaining_time--;
        total_time++;

        if (processes[next_process].remaining_time == 0) {
            completed++;
            processes[next_process].turnaround_time = total_time - processes[next_process].arrival_time;
            processes[next_process].waiting_time = processes[next_process].turnaround_time - processes[next_process].burst_time;
        }
    }

    double total_turnaround_time = 0;
    double total_waiting_time = 0;

    printf("Process\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time, processes[i].waiting_time);

        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;
    }

    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
```

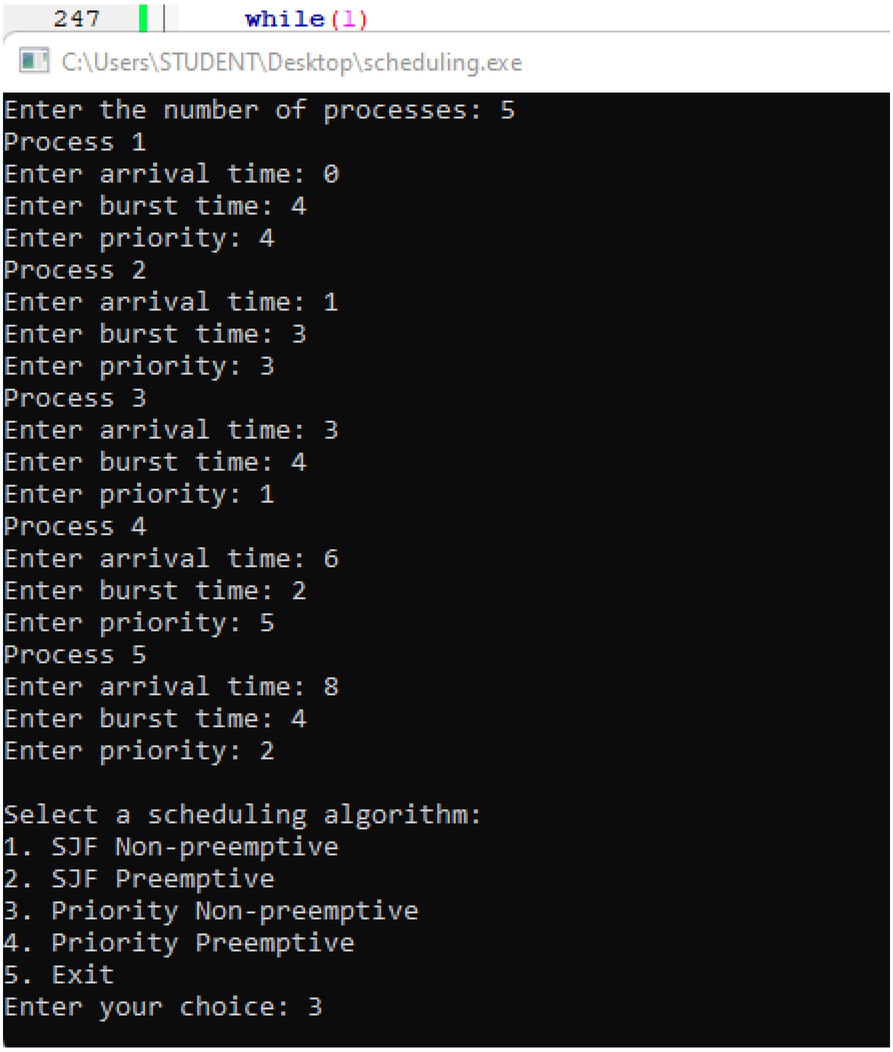
```
}

void priority_nonpreemptive(struct Process processes[], int n) {
   int i,j,count=0,m;
   for(i=0;i<n;i++)
   {
   if(processes[i].arrival_time==0)
   count++;
}
if(count==n||count==1)
{
if(count==n)
{
for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (processes[j].priority > processes[j + 1].priority) {
            struct Process temp = processes[j];
            processes[j] = processes[j + 1];
            processes[j + 1] = temp;
         }
      }
   }
}

else
{
   for (i = 1; i < n - 1; i++) {
      for (j = 1; j <= n - i - 1; j++) {
         if (processes[j].priority > processes[j + 1].priority) {
            struct Process temp = processes[j];
            processes[j] = processes[j + 1];
```

```
                processes[j + 1] = temp;
            }
        }
    }
}
}

    int total_time = 0;
    double total_turnaround_time = 0;
    double total_waiting_time = 0;

    for (i = 0; i < n; i++) {
        total_time += processes[i].burst_time;
        processes[i].turnaround_time = total_time - processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;

        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;
    }

    printf("Process\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time,
processes[i].waiting_time);
    }

    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
}

void priority_preemptive(struct Process processes[], int n) {
    int total_time = 0,i;
    int completed = 0;

    while (completed < n) {
        int highest_priority = -1;
        int next_process = -1;

        for (i = 0; i < n; i++) {
            if (processes[i].arrival_time <= total_time && processes[i].remaining_time > 0) {
                if (highest_priority == -1 || processes[i].priority < highest_priority) {
                    highest_priority = processes[i].priority;
                    next_process = i;
                }
            }
        }

        if (next_process == -1) {
            total_time++;
            continue;
        }

        processes[next_process].remaining_time--;
        total_time++;

        if (processes[next_process].remaining_time == 0) {
            completed++;
            processes[next_process].turnaround_time = total_time - processes[next_process].arrival_time;
            processes[next_process].waiting_time = processes[next_process].turnaround_time - processes[next_process].burst_time;
        }
    }
    double total_turnaround_time = 0;
    double total_waiting_time = 0;

    printf("Process\tTurnaround Time\tWaiting Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time,
processes[i].waiting_time);

        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;
    }

    printf("Average Turnaround Time: %.2f\n", total_turnaround_time / n);
    printf("Average Waiting Time: %.2f\n", total_waiting_time / n);
}

int main() {
    int n, quantum,i,choice;
    struct Process processes[MAX_PROCESSES];

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

    for (i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
        printf("Enter arrival time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Enter burst time: ");
        scanf("%d", &processes[i].burst_time);
        printf("Enter priority: ");
        scanf("%d", &processes[i].priority);
        processes[i].pid = i + 1;
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].turnaround_time = 0;
        processes[i].waiting_time = 0;
    }
    while(1)
    {
         printf("\nSelect a scheduling algorithm:\n");
    printf("1. SJF Non-preemptive\n");
    printf("2. SJF Preemptive\n");
    printf("3. Priority Non-preemptive\n");
    printf("4. Priority Preemptive\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("\nSJF Non-preemptive Scheduling:\n");
            sjf_nonpreemptive(processes, n);
            break;
        case 2:
            printf("\nSJF Preemptive Scheduling:\n");
            sjf_preemptive(processes, n);
            break;
        case 3:
            printf("\nPriority Non-preemptive Scheduling:\n");
            priority_nonpreemptive(processes, n);
            break;
        case 4:
            printf("\nPriority Preemptive Scheduling:\n");
            priority_preemptive(processes, n);
            break;
        case 5:exit(0);
            break;
        default:
            printf("Invalid choice!\n");
            return 1;
    }
    }

    return 0;
}
```



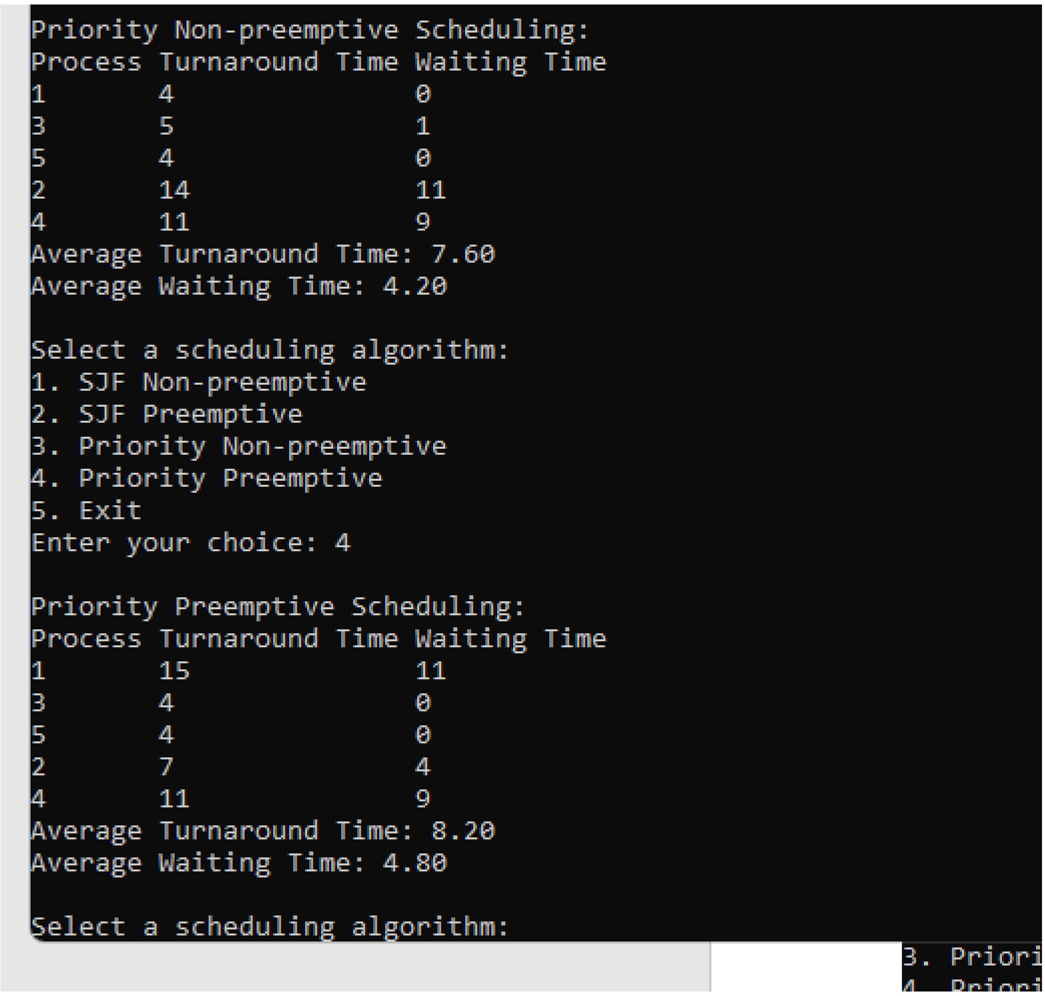
C:\Users\STUDENT\Desktop\scheduling.exe

```
Process 5
Enter arrival time: 8
Enter burst time: 4
Enter priority: 2

Select a scheduling algorithm:
1. SJF Non-preemptive
2. SRTF
3. Priority Non-preemptive
4. Priority Preemptive
5. Exit
Enter your choice: 2

SJF Preemptive Scheduling:
Process Turnaround Time Waiting Time
1       4               0
2       6               3
3       10              6
4       3               1
5       9               5
Average Turnaround Time: 6.40
Average Waiting Time: 3.00

Select a scheduling algorithm:
1. SJF Non-preemptive
2. SRTF
3. Priority Non-preemptive
4. Priority Preemptive
5. Exit
```



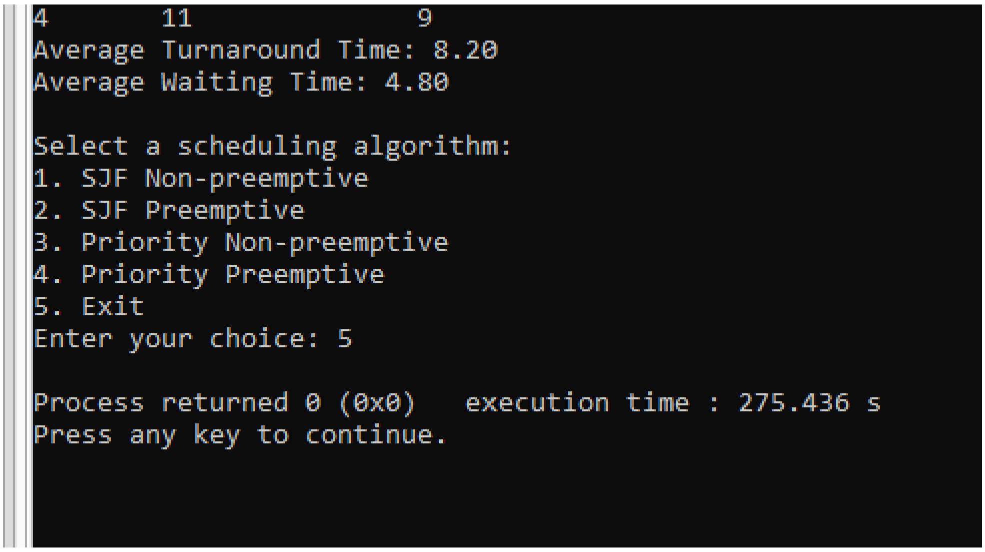
```
	while(1)
```

```
Enter the number of processes: 5
Process 1
Enter arrival time: 0
Enter burst time: 4
Enter priority: 4
Process 2
Enter arrival time: 1
Enter burst time: 3
Enter priority: 3
Process 3
Enter arrival time: 3
Enter burst time: 4
Enter priority: 1
Process 4
Enter arrival time: 6
Enter burst time: 2
Enter priority: 5
Process 5
Enter arrival time: 8
Enter burst time: 4
Enter priority: 2

Select a scheduling algorithm:
1. SJF Non-preemptive
2. SJF Preemptive
3. Priority Non-preemptive
4. Priority Preemptive
5. Exit
Enter your choice: 3
```



```
Priority Non-preemptive Scheduling:
Process Turnaround Time Waiting Time
1       4               0
3       5               1
5       4               0
2       14              11
4       11              9
Average Turnaround Time: 7.60
Average Waiting Time: 4.20

Select a scheduling algorithm:
1. SJF Non-preemptive
2. SJF Preemptive
3. Priority Non-preemptive
4. Priority Preemptive
5. Exit
Enter your choice: 4

Priority Preemptive Scheduling:
Process Turnaround Time Waiting Time
1       15              11
3       4               0
5       4               0
2       7               4
4       11              9
Average Turnaround Time: 8.20
Average Waiting Time: 4.80

Select a scheduling algorithm:
```

```
3. Priori
4. Priori
```



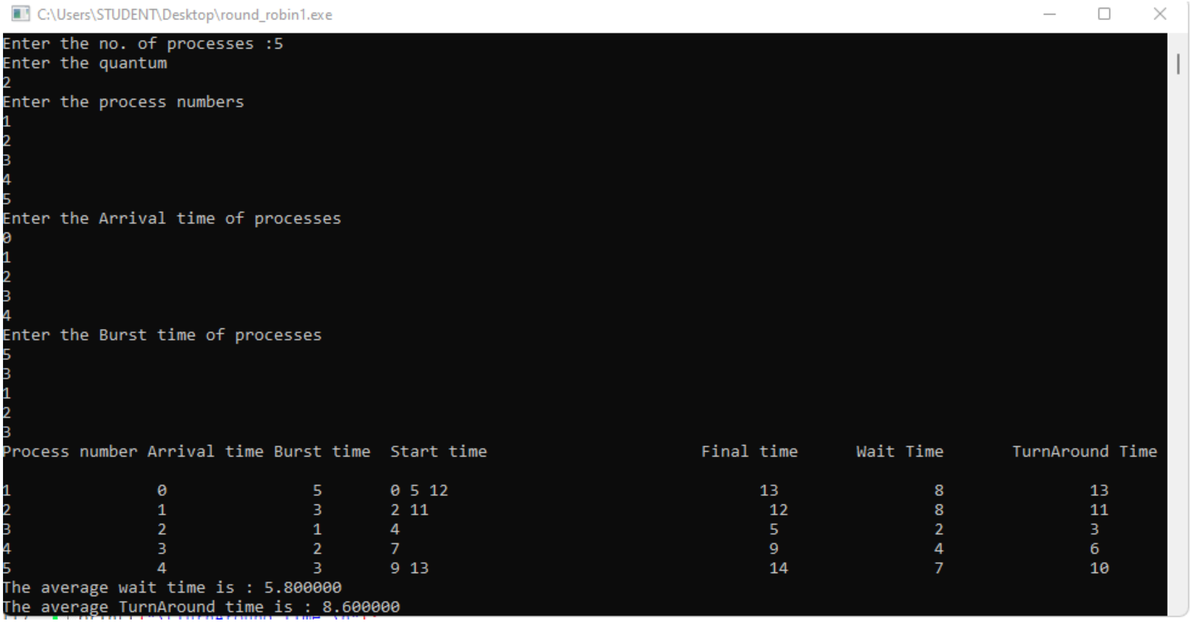
```
4       11              9
Average Turnaround Time: 8.20
Average Waiting Time: 4.80

Select a scheduling algorithm:
1. SJF Non-preemptive
2. SJF Preemptive
3. Priority Non-preemptive
4. Priority Preemptive
5. Exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 275.436 s
Press any key to continue.
```

Round Robin:

```
#include<stdio.h>

#include<limits.h>

#include<stdbool.h>


struct P{

int AT,BT,ST[20],WT,FT,TAT,pos;

};


int quant;

int main(){

int n,i,j;

printf("Enter the no. of processes :");

scanf("%d",&n);

struct P p[n];


printf("Enter the quantum  \n");

scanf("%d",&quant);


printf("Enter the process numbers \n");

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

scanf("%d",&(p[i].pos));


printf("Enter the Arrival time of processes \n");

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

scanf("%d",&(p[i].AT));


printf("Enter the Burst time of processes \n");

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

scanf("%d",&(p[i].BT));
```

```
int c=n,s[n][20];

float time=0,mini=INT_MAX,b[n],a[n];



int index=-1;

for(i=0;i<n;i++){

        b[i]=p[i].BT;

        a[i]=p[i].AT;

        for(j=0;j<20;j++){

        s[i][j]=-1;

        }

}


int tot_wt,tot_tat;

tot_wt=0;

tot_tat=0;

bool flag=false;


while(c!=0){


mini=INT_MAX;

flag=false;


for(i=0;i<n;i++){

        float p=time+0.1;

        if(a[i]<=p && mini>a[i] && b[i]>0){

        index=i;

        mini=a[i];
        flag=true;

        }
}

if(!flag){
        time++;
        continue;
}

j=0;

while(s[index][j]!=-1){
j++;
}

if(s[index][j]==-1){
s[index][j]=time;
p[index].ST[j]=time;
}

if(b[index]<=quant){
time+=b[index];
b[index]=0;
}
else{
time+=quant;
b[index]-=quant;
}

if(b[index]>0){
```

```
a[index]=time+0.1;

}


if(b[index]==0){

c--;

p[index].FT=time;

p[index].WT=p[index].FT-p[index].AT-p[index].BT;

tot_wt+=p[index].WT;

p[index].TAT=p[index].BT+p[index].WT;

tot_tat+=p[index].TAT;


}


}

printf("Process number ");

printf("Arrival time ");

printf("Burst time ");

printf("\tStart time");

j=0;

while(j!=10){

j+=1;

printf(" ");

}

printf("\t\tFinal time");

printf("\tWait Time ");

printf("\tTurnAround Time \n");



for(i=0;i<n;i++){

printf("%d \t\t",p[i].pos);

printf("%d \t\t",p[i].AT);
printf("%d \t",p[i].BT);
j=0;
int v=0;
while(s[i][j]!=-1){
printf("%d ",p[i].ST[j]);
j++;
v+=3;
}
while(v!=40){
printf(" ");
v+=1;
}
printf("%d \t\t",p[i].FT);
printf("%d \t\t",p[i].WT);
printf("%d \n",p[i].TAT);


}


double avg_wt,avg_tat;
avg_wt=tot_wt/(float)n;
avg_tat=tot_tat/(float)n;


printf("The average wait time is : %lf\n",avg_wt);
printf("The average TurnAround time is : %lf\n",avg_tat);


return 0;
}
```



```
Enter the no. of processes :5
Enter the quantum
2
Enter the process numbers
1
2
3
4
5
Enter the Arrival time of processes
0
1
2
3
4
Enter the Burst time of processes
5
3
1
2
3
Process number Arrival time Burst time  Start time                      Final time      Wait Time       TurnAround Time

1               0               5       0 5 12                                 13               8               13
2               1               3       2 11                                    12              8               11
3               2               1       4                                       5               2               3
4               3               2       7                                       9               4               6
5               4               3       9 13                                    14              7               10
The average wait time is : 5.800000
The average TurnAround time is : 8.600000
```