LP-I Assignment No. 1[A]

Name : Ravindra Dayaram Bagul
Roll No. : C31105
Batch : T-1

Problem Statement : To write a C++ program to implement FCFS (Non Preemptive) algorithm of CPU scheduling.

Input

```
#include <iostream>
#include <algorithm>
#include <iomanip>
using namespace std;

struct process {
   int pid;
   int arrival_time;
   int burst_time;
   int start_time;
   int completion_time;
   int turnaround_time;
   int waiting_time;
   int response_time;
};

bool compareArrival(process p1, process p2)
{
   return p1.arrival_time < p2.arrival_time;
}

bool compareID(process p1, process p2)
{
   return p1.pid < p2.pid;
}

int main() {

   int n;
   struct process p[100];
   float avg_turnaround_time;
   float avg_waiting_time;
   float avg_response_time;
   int total_turnaround_time = 0;
   int total_waiting_time = 0;
   int total_response_time = 0;
   int total_idle_time = 0;
```

```
    cout << setprecision(2) << fixed;

    cout<<"Enter the number of processes: ";
    cin>>n;

    for(int i = 0; i < n; i++) {
        cout<<"Enter arrival time of process "<<i+1<<": ";
        cin>>p[i].arrival_time;
        cout<<"Enter burst time of process "<<i+1<<": ";
        cin>>p[i].burst_time;
        p[i].pid = i+1;
        cout<<endl;
    }

    sort(p,p+n,compareArrival);

    for(int i = 0; i < n; i++)
{
        p[i].start_time = (i == 0)?p[i].arrival_time:max(p[i-1].completion_time,
p[i].arrival_time);
        p[i].completion_time = p[i].start_time + p[i].burst_time;
        p[i].turnaround_time = p[i].completion_time - p[i].arrival_time;
        p[i].waiting_time = p[i].turnaround_time - p[i].burst_time;
        p[i].response_time = p[i].start_time - p[i].arrival_time;

        total_turnaround_time += p[i].turnaround_time;
        total_waiting_time += p[i].waiting_time;
        total_response_time += p[i].response_time;
        total_idle_time += (i == 0) ?(p[i].arrival_time):(p[i].start_time -p[i-
1].completion_time);
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;

    sort(p,p+n,compareID);

    cout<<endl;
    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT
\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++)
   {
        cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<
p[i].start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"
<<p[i].waiting_time<<"\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
    }
```

```
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;

}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:

TAT = CT - AT
WT = TAT - BT
RT = ST - AT

*/
```

Output

```
Enter the number of processes: 4
Enter arrival time of process 1: 0
Enter burst time of process 1: 2

Enter arrival time of process 2: 1
Enter burst time of process 2: 2

Enter arrival time of process 3: 5
Enter burst time of process 3: 3

Enter arrival time of process 4: 6
Enter burst time of process 4: 4


#P    AT    BT    ST    CT    TAT    WT    RT

1     0     2     0     2     2      0     0

2     1     2     2     4     3      1     1

3     5     3     5     8     3      0     0

4     6     4     8     12    6      2     2
```

Average Turnaround Time = 3.50
Average Waiting Time = 0.75
Average Response Time = 0.75

LP-I Assignment No. 1[B]

Name : Ravindra Dayaram Bagul
Roll No. : C31105
Batch : T-1

Problem Statement : To write a C++ program to implement SJF (Non Preemptive) algorithm of CPU scheduling.

Input

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string.h>
using namespace std;

struct process {
   int pid;
   int arrival_time;
   int burst_time;
   int start_time;
   int completion_time;
   int turnaround_time;
   int waiting_time;
   int response_time;
};

int main() {

   int n;
   struct process p[100];
   float avg_turnaround_time;
   float avg_waiting_time;
   float avg_response_time;
   int total_turnaround_time = 0;
   int total_waiting_time = 0;
   int total_response_time = 0;
   int total_idle_time = 0;
   int is_completed[100];
   memset(is_completed,0,sizeof(is_completed));

   cout << setprecision(2) << fixed;

   cout<<"Enter the number of processes: ";
   cin>>n;

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

```
    cout<<"Enter arrival time of process "<<i+1<<": ";
    cin>>p[i].arrival_time;
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    p[i].pid = i+1;
    cout<<endl;
}

int current_time = 0;
int completed = 0;
int prev = 0;

while(completed != n) {
    int idx = -1;
    int mn = 10000000;
    for(int i = 0; i < n; i++) {
        if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
            if(p[i].burst_time < mn) {
                mn = p[i].burst_time;
                idx = i;
            }
            if(p[i].burst_time == mn) {
                if(p[i].arrival_time < p[idx].arrival_time) {
                    mn = p[i].burst_time;
                    idx = i;
                }
            }
        }
    }

    if(idx != -1)
    {
        p[idx].start_time = current_time;
        p[idx].completion_time = p[idx].start_time + p[idx].burst_time;
        p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
        p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
        p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

        total_turnaround_time += p[idx].turnaround_time;
        total_waiting_time += p[idx].waiting_time;
        total_response_time += p[idx].response_time;
        total_idle_time += p[idx].start_time - prev;

        is_completed[idx] = 1;
        completed++;
        current_time = p[idx].completion_time;
        prev = current_time;
    }
```

```
 else {
          current_time++;
       }

   }

   avg_turnaround_time = (float) total_turnaround_time / n;
   avg_waiting_time = (float) total_waiting_time / n;
   avg_response_time = (float) total_response_time / n;

   cout<<endl<<endl;

   cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT
\t"<<"\n"<<endl;

   for(int i = 0; i < n; i++)
{
      cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<
p[i].start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"
<<p[i].waiting_time<<"\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
}
   cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
   cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
   cout<<"Average Response Time = "<<avg_response_time<<endl;

}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:

TAT = CT - AT
WT = TAT - BT
RT = ST - AT

*/
```

Output

Enter the number of processes: 4
Enter arrival time of process 1: 1
Enter burst time of process 1: 3

Enter arrival time of process 2: 2
Enter burst time of process 2: 4

Enter arrival time of process 3: 1
Enter burst time of process 3: 2

Enter arrival time of process 4: 4
Enter burst time of process 4: 4

#P	AT	BT	ST	CT	TAT	WT	RT
1	1	3	3	6	5	2	2
2	2	4	6	10	8	4	4
3	1	2	1	3	2	0	0
4	4	4	10	14	10	6	6

Average Turnaround Time = 6.25
Average Waiting Time = 3.00
Average Response Time = 3.00

LP-I Assignment No. 1[C]

Name : Ravindra Dayaram Bagul
Roll No. : C31105
Batch : T-1

Problem Statement : To write a C++ program to implement Priority (Non-Preemptive) algorithm of CPU scheduling.

Input

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string.h>
using namespace std;

struct process {
   int pid;
   int arrival_time;
   int burst_time;
   int priority;
   int start_time;
   int completion_time;
   int turnaround_time;
   int waiting_time;
   int response_time;
};

int main() {

   int n;
   struct process p[100];
   float avg_turnaround_time;
   float avg_waiting_time;
   float avg_response_time;
   int total_turnaround_time = 0;
   int total_waiting_time = 0;
   int total_response_time = 0;
   int total_idle_time = 0;
   int is_completed[100];
   memset(is_completed,0,sizeof(is_completed));

   cout << setprecision(2) << fixed;

   cout<<"Enter the number of processes: ";
   cin>>n;

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

```
    cout<<"Enter arrival time of process "<<i+1<<": ";
    cin>>p[i].arrival_time;
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    cout<<"Enter priority of the process "<<i+1<<": ";
    cin>>p[i].priority;
    p[i].pid = i+1;
    cout<<endl;
}

int current_time = 0;
int completed = 0;
int prev = 0;

while(completed != n) {
    int idx = -1;
    int mx = -1;
    for(int i = 0; i < n; i++) {
        if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
            if(p[i].priority > mx) {
                mx = p[i].priority;
                idx = i;
            }
            if(p[i].priority == mx) {
                if(p[i].arrival_time < p[idx].arrival_time) {
                    mx = p[i].priority;
                    idx = i;
                }
            }
        }
    }
    if(idx != -1) {
        p[idx].start_time = current_time;
        p[idx].completion_time = p[idx].start_time + p[idx].burst_time;
        p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
        p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
        p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

        total_turnaround_time += p[idx].turnaround_time;
        total_waiting_time += p[idx].waiting_time;
        total_response_time += p[idx].response_time;
        total_idle_time += p[idx].start_time - prev;

        is_completed[idx] = 1;
        completed++;
        current_time = p[idx].completion_time;
        prev = current_time;
    }
```

```
        else {
            current_time++;
        }

    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;
    cout<<endl<<endl;

    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"W
T\t"<<"RT\t"<<"\n"<<endl;

    for(int i = 0; i < n; i++) {
        cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].
priority<<"\t"<<p[i].start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaro
und_time<<"\t"<<p[i].waiting_time<<"\t"<<p[i].response_time<<"\t"<<"\n"<<en
dl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;

}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:

TAT = CT - AT
WT = TAT - BT
RT = ST - AT

*/
```

Output

Enter the number of processes: 6
Enter arrival time of process 1: 0
Enter burst time of process 1: 4
Enter priority of the process 1: 4

Enter arrival time of process 2: 1
Enter burst time of process 2: 5
Enter priority of the process 2: 5

Enter arrival time of process 3: 2
Enter burst time of process 3: 1
Enter priority of the process 3: 7

Enter arrival time of process 4: 3
Enter burst time of process 4: 2
Enter priority of the process 4: 2

Enter arrival time of process 5: 4
Enter burst time of process 5: 3
Enter priority of the process 5: 1

Enter arrival time of process 6: 5
Enter burst time of process 6: 6
Enter priority of the process 6: 6

#P	AT	BT	PRI	ST	CT	TAT	WT	RT
1	0	4	4	0	4	4	0	0
2	1	5	5	11	16	15	10	10
3	2	1	7	4	5	3	2	2
4	3	2	2	16	18	15	13	13
5	4	3	1	18	21	17	14	14
6	5	6	6	5	11	6	0	0

Average Turnaround Time = 10.00
Average Waiting Time = 6.50
Average Response Time = 6.50

LP-I Assignment No. 1[D]

Name : Ravindra Dayaram Bagul
Roll No. : C31105
Batch : T-1

Problem Statement : To write a Java program to implement Round Robin(Preemptive) algorithm of CPU scheduling.

Input

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include<cstring>
#include <queue>
using namespace std;

struct process {
   int pid;
   int arrival_time;
   int burst_time;
   int start_time;
   int completion_time;
   int turnaround_time;
   int waiting_time;
   int response_time;
};

bool compare1(process p1, process p2)
{
   return p1.arrival_time < p2.arrival_time;
}

bool compare2(process p1, process p2)
{
   return p1.pid < p2.pid;
}

int main() {

   int n;
   int tq;
   struct process p[100];
   float avg_turnaround_time;
   float avg_waiting_time;
   float avg_response_time;
   int total_turnaround_time = 0;
   int total_waiting_time = 0;
```

```
int total_response_time = 0;
int total_idle_time = 0;
int burst_remaining[100];
int idx;

cout << setprecision(2) << fixed;

cout<<"Enter the number of processes: ";
cin>>n;
cout<<"Enter time quantum: ";
cin>>tq;

for(int i = 0; i < n; i++) {
   cout<<"Enter arrival time of process "<<i+1<<": ";
   cin>>p[i].arrival_time;
   cout<<"Enter burst time of process "<<i+1<<": ";
   cin>>p[i].burst_time;
   burst_remaining[i] = p[i].burst_time;
   p[i].pid = i+1;
   cout<<endl;
}

sort(p,p+n,compare1);

queue<int> q;
int current_time = 0;
q.push(0);
int completed = 0;
int mark[100];
memset(mark,0,sizeof(mark));
mark[0] = 1;

while(completed != n) {
   idx = q.front();
   q.pop();

   if(burst_remaining[idx] == p[idx].burst_time) {
      p[idx].start_time = max(current_time,p[idx].arrival_time);
      total_idle_time += p[idx].start_time - current_time;
      current_time = p[idx].start_time;
   }

   if(burst_remaining[idx]-tq > 0) {
      burst_remaining[idx] -= tq;
      current_time += tq;
   }
   else {
      current_time += burst_remaining[idx];
```

```
            burst_remaining[idx] = 0;
            completed++;

            p[idx].completion_time = current_time;
            p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
            p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
            p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

            total_turnaround_time += p[idx].turnaround_time;
            total_waiting_time += p[idx].waiting_time;
            total_response_time += p[idx].response_time;
        }

        for(int i = 1; i < n; i++) {
            if(burst_remaining[i] > 0 && p[i].arrival_time <= current_time && mark[i]
== 0) {
                q.push(i);
                mark[i] = 1;
            }
        }
        if(burst_remaining[idx] > 0) {
            q.push(idx);
        }

        if(q.empty()) {
            for(int i = 1; i < n; i++) {
                if(burst_remaining[i] > 0) {
                    q.push(i);
                    mark[i] = 1;
                    break;
                }
            }
        }
    }

    avg_turnaround_time = (float) total_turnaround_time / n;
    avg_waiting_time = (float) total_waiting_time / n;
    avg_response_time = (float) total_response_time / n;

    sort(p,p+n,compare2);

    cout<<endl;
    cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT
\t"<<"\n"<<endl;

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

```
        cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].
start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[
i].waiting_time<<"\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
    }
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
    cout<<"Average Response Time = "<<avg_response_time<<endl;
}

/*

AT - Arrival Time of the process
BT - Burst time of the process
ST - Start time of the process
CT - Completion time of the process
TAT - Turnaround time of the process
WT - Waiting time of the process
RT - Response time of the process

Formulas used:

TAT = CT - AT
WT = TAT - BT
RT = ST - AT

*/
```

Output

Enter the number of processes: 4

Enter time quantum: 2

Enter arrival time of process 1: 0
Enter burst time of process 1: 5

Enter arrival time of process 2: 1
Enter burst time of process 2: 4

Enter arrival time of process 3: 2
Enter burst time of process 3: 2

Enter arrival time of process 4: 4
Enter burst time of process 4: 1

#P	AT	BT	ST	CT	TAT	WT	RT
1	0	5	0	12	12	7	0
2	1	4	2	11	10	6	1
3	2	2	4	6	4	2	2
4	4	1	8	9	5	4	4

Average Turnaround Time = 7.75
Average Waiting Time = 4.75
Average Response Time = 1.75