```
#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;
	float cpu_utilisation;
	int total_turnaround_time = 0;
```

```
int total_waiting_time = 0;

int total_response_time = 0;

int total_idle_time = 0;

float throughput;

int burst_remaining[100];

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;

	burst_remaining[i] = p[i].burst_time;

	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(burst_remaining[i] < mn) {
                mn = burst_remaining[i];
                idx = i;
            }
            if(burst_remaining[i] == mn) {
                if(p[i].arrival_time < p[idx].arrival_time) {
                    mn = burst_remaining[i];
                    idx = i;
                }
            }
        }
    }

    if(idx != -1) {
        if(burst_remaining[idx] == p[idx].burst_time) {
            p[idx].start_time = current_time;
            total_idle_time += p[idx].start_time - prev;
        }
        burst_remaining[idx] -= 1;
        current_time++;
```

```
            prev = current_time;

            if(burst_remaining[idx] == 0) {
                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;

                is_completed[idx] = 1;
                completed++;
            }
        }
        else {
             current_time++;
        }
    }

    int min_arrival_time = 10000000;
    int max_completion_time = -1;
    for(int i = 0; i < n; i++) {
        min_arrival_time = min(min_arrival_time,p[i].arrival_time);
        max_completion_time = max(max_completion_time,p[i].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;
    cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_completion_time )*100;
    throughput = float(n) / (max_completion_time - min_arrival_time);

    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].respons
e_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;
    cout<<"CPU Utilization = "<<cpu_utilisation<<"%"<<endl;
    cout<<"Throughput = "<<throughput<<" process/unit 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

*/