Implement the First Come First Serve CPU scheduling algorithm by reading data from file.

Inputs:

The program will read from an input file containing a list of processes along with other data require for

scheduling.

The input file will look like this:

P1 0 20.0

P2 2 15.0

P3 6 27.0

p4 4 36.0

- **The file containing the information on the processes will have each process on a separate line. The**

processes will be in the file in the order in which they arrive at the OS.

- **Each line will have a process name that will be a string.**
- **Following the name will be the arrival time of the process**
- **Following arrival time will be the total burst time.**

Outputs:

The program must print out the time taken by each process to complete (turnaround time) and the wait

time and compute the average turnaround time for all processes.

```
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include<string>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <iomanip>
using namespace std;

//Class process for storing information about every
process

class process
{

//Instance variables

private:
    string name;
    int process_id;
    int arrival_time;
    int burst_time;
    int completion_time;
    int turn_around_time;
    int wait_time;
public:

//Constructor for assigning the values to instance
variable

process(string name,int arrival_time,int burst_time,int
process_id)
    {
        this->name=name;
        this->arrival_time=arrival_time;
        this->burst_time=burst_time;
        this->process_id=process_id;
        this->completion_time=0;
        this->turn_around_time=0;
        this->wait_time=0;
```

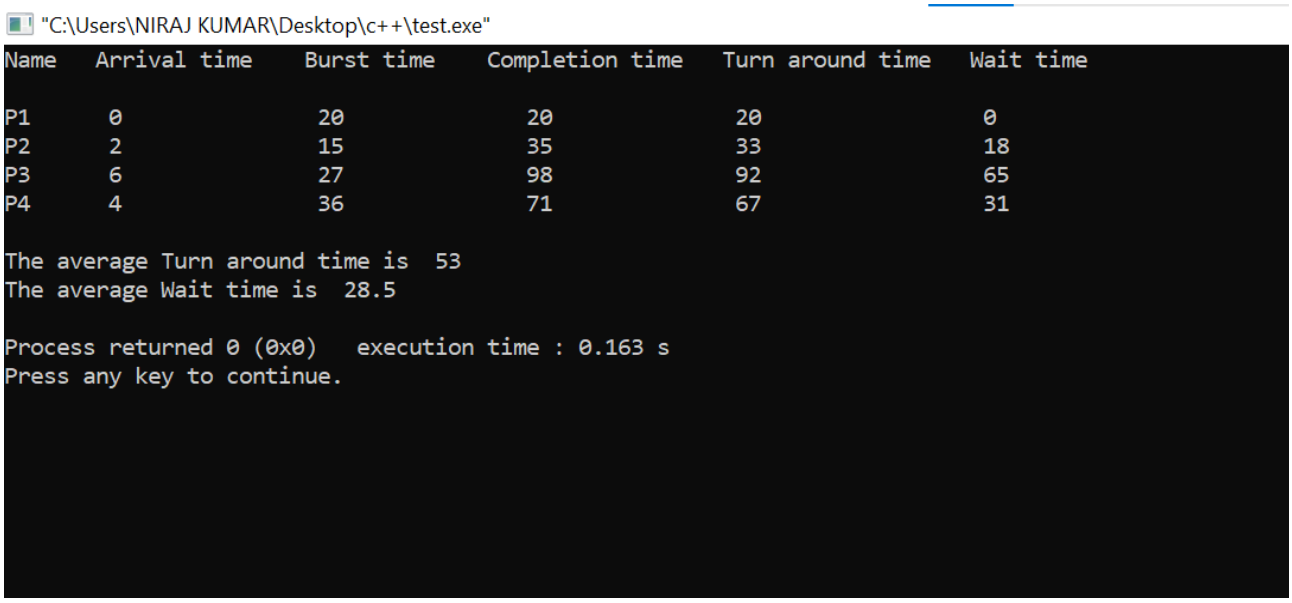
```
    }

    void display_details()
    {
        cout<<name<<"\t"<<arrival_time<<"\t\t"
            <<burst_time<<"\t\t"<<completion_time<<"\t\t"
            <<turn_around_time<<"\t\t
"<<wait_time<<endl;
    }

    int get_turn_around_time()
    {
        return turn_around_time;
    }

    int get_wait_time()
    {
        return wait_time;
    }

//Friend functions that will access private members of
class

    friend void calculate_completion_time(process **);
    friend void calculate_turn_around_time(process **);
    friend void calculate_wait_time(process **);
    friend bool compareArrival(process *, process *);
    friend bool compareID(process *, process *);


};

//Comparator function based on arrival time

bool compareArrival(process *p1, process *p2)
{
    return p1->arrival_time < p2->arrival_time;
}

//Comparator function based on process id

bool compareID(process *p1, process *p2)
```

```
{
    return p1->process_id < p2->process_id;
}

//Calculating completion time for each process

void calculate_completion_time(process **p)
{
    int time_counter=0;
    sort(p,p+4,compareArrival);
    for (int i=0;i<=3;i++)
    {
        if(p[i]->arrival_time <= time_counter)
        {
            time_counter=time_counter + p[i]->burst_time;
            p[i]->completion_time=time_counter;
        }
        else
        {
            time_counter=p[i]->arrival_time;
            time_counter=time_counter + p[i]->burst_time;
            p[i]->completion_time=time_counter;
        }
    }
}

//Calculating turn around time

void calculate_turn_around_time(process **p)
{
    for (int i=0;i<=3;i++)
        p[i]->turn_around_time=p[i]->completion_time -
p[i]->arrival_time;
}

//Calculating wait time

void calculate_wait_time(process **p)
{
    for (int i=0;i<=3;i++)
        p[i]->wait_time=p[i]->turn_around_time - p[i]-
>burst_time;
```

```
}

int main()
{
    int at[4]; //arrival time array for each process
    int bt[4]; //burst time array for each process

     string line[4]; Reading Lines from files
     ifstream fio;
     fio.open("C:\\Users\\NIRAJ
KUMAR\\Desktop\\c++\\Input.txt");
     for (int i=0;i<=3;i++)
    {
    getline(fio,line[i]);

     //for converting string to integer , using
stringstream class

    stringstream geek1(line[i].substr(3,1));
     geek1 >> at[i];
     stringstream geek2(line[i].substr(5,4));
     geek2 >> bt[i];
    }


     fio.close();

//creating objects for process

    process **p=(process**)malloc(sizeof(process*)*4);
    p[0]=new process("P1",at[0],bt[0],1);
    p[1]=new process("P2",at[1],bt[1],2);
    p[2]=new process("P3",at[2],bt[2],3);
    p[3]=new process("P4",at[3],bt[3],4);

    calculate_completion_time(p);
    sort(p,p+4,compareID);
    calculate_turn_around_time(p);
    calculate_wait_time(p);
    cout<<"Name   Arrival time    Burst time    Completion
time   Turn around time   Wait time\n\n";
```

```
//printing the details for each process

    for (int i=0;i<=3;i++)
        p[i]->display_details();

//calculating average wait time and turn around time

    double total_wait_time=0,total_turn_around_time=0;
    for (int i=0;i<=3;i++)
    {
     total_turn_around_time += p[i]-
>get_turn_around_time();
     total_wait_time += p[i]->get_wait_time();
    }

    cout<<"\nThe average Turn around time is
"<<total_turn_around_time/4<<endl;
    cout<<"The average Wait time is
"<<total_wait_time/4<<endl;
}
```

Output



Input file


Input - Notepad
File Edit Format View Help
P1 0 20.0
P2 2 15.0
P3 6 27.0
P4 4 36.0