```
//OS Assignment CPU process scheduling
//Submitted by: Sanskar sharma
//PRN: 0120180381
//Roll number: 090 TY
#include<iostream>
#include<string.h>

using namespace std;

class processnode
{
    public:
        //user input
        char p_name[50];
        double a_time=0;
        double b_time=NULL;
        //Evaluation
        double s_time=NULL;
        double w_time=0;
        double ta_time=NULL;


};

class process_scheduling
{
    public:
        void input_processes();
        void FCFS(int n, processnode process_arr[]);
        void SJF(int n, processnode process_arr[]);
        void Roundrobin(int n, processnode process_arr[]);


};

/*class node
{
    public:
        int index;
        node*next;//pointer of a node
};

//Queue for Creating ready queue in RR algorithm.
class queue
{
    public:
        node *front,*rear;
        queue()
        {
            front=NULL;
            rear=NULL;
        }
        void insert(int val)
        {
            node*temp;
            temp=new node();
            if(temp==NULL)
            {
                cout<<"Error in memory allocation";exit(-1);
            }
            temp->index=val;
            temp->next=NULL;
            if(rear==NULL)
            {
                rear=temp;
                front=temp;
            }
            rear->next=temp;
            rear=temp;

        }
        int delete_()
        {

            if(queue_empty()==1)
            {
                return -1;

            }
            else
            {
                int val;
                val=front->index;
                node*temp=front;
                front=front->next;
                if(front==NULL)
                {
                    rear=NULL;
                }

                delete(temp);
                return val;
            }
        }
        int queue_empty()
        {
            if(front==NULL)
            {
                return 1;
            }
            else
            {
                return 0;
            }
        }
};*/

void sorting(int n, processnode *process_arr)
{
    processnode temp;
    for(int i=0; i<n; ++i)
    {
        for(int j=0; j<n-1; ++j)
        {
            if(process_arr[j].a_time>process_arr[j+1].a_time)
            {
                temp=process_arr[j];
                process_arr[j]=process_arr[j+1];
                process_arr[j+1]=temp;
            }
        }
    }
```

```
}

//This function returns index of the shortest burst time of the arrived and not completed processes.
int shortes_b_time(int n, int arrived[], int completed[], processnode process_arr[])
{
    double min=9999; //initialising minimum as largest number! say 9999
    int index=-1;
    for(int i=0;i<n;++i)
    {
        if(arrived[i]==1)//process arrived
        {
            if(completed[i]!=1)//Processes incompleted
            {
                if(process_arr[i].b_time<min)
                {
                    min=process_arr[i].b_time;
                    index=i;
                }
            }
        }
    }
    return index;
}

/*
int check_arrived(int n,double time,processnode process_arr[],int arrived[])
{
    int flag=0;
    for(int i=0;i<n;++i)
    {
        if(arrived[i]!=1)
        {
            if(process_arr[i].a_time<=time)
            {
                flag=1;
                break;
            }
        }
    }
    return flag;
}*/


void process_scheduling::Roundrobin(int n, processnode process_arr[])
{
    //round robin algorithm.
    double time_quantum,time,ta_time_avg=0, w_time_avg=0;
    cout<<"\n\t\tEnter the time slice/quantum for RR algorithm: ";
    cin>>time_quantum;
    int remain=n,i,temps=0;
    sorting(n,process_arr);
    double b_time_arr[n];
    for(int i=0;i<n;++i)
    {
        b_time_arr[i]=process_arr[i].b_time;
    }
    cout<<"*******************Round Robin Algorithm*******************";
    cout<<"\n\n\tProcess:  \t:Turnaround Time: Waiting Time\n\n";
    for(time=0,i=0;remain!=0;)
    {
        if(b_time_arr[i]<=time_quantum && b_time_arr[i]>0)
        {
            time += b_time_arr[i];
            //Addition using shorthand operators
            b_time_arr[i]=0;
            temps=1;
        }

        else if(b_time_arr[i]>0)
        {
            b_time_arr[i] -= time_quantum;
            //Subtraction using shorthand operators
            time += time_quantum;
            //Addition using shorthand operators
        }

        if(b_time_arr[i]==0 && temps==1)
        {
            remain--;
            //Desplaying the result of wating, turn around time:
            process_arr[i].ta_time=time-process_arr[i].a_time;
            process_arr[i].w_time=time-process_arr[i].a_time-process_arr[i].b_time;
            cout<<"\tProcess: "<<process_arr[i].p_name<<"\t:\t"<<process_arr[i].ta_time;
            cout<<"\t:\t"<<process_arr[i].w_time;
            //printf("Process{%d}\t:\t%d\t:\t%d\n",i+1,process_arr[i].ta_time,process_arr[i].w_time);
            cout<<endl;

            w_time_avg += time-process_arr[i].a_time-process_arr[i].b_time;
            ta_time_avg += time-process_arr[i].a_time;
            temps=0;
        }

        if(i == n-1)
            i=0;
        else if(process_arr[i+1].a_time <= time)
            i++;
        else
            i=0;
    }
    cout<<"\n\t\t\tAverage Waiting time is: "<<w_time_avg/n;
    cout<<"\n\t\t\tAverage Turn Around time is: "<<ta_time_avg/n;

}

/*
void process_scheduling::Roundrobin(int n, processnode process_arr[])
{
    //Curren status incomplete
    //using queue to show the gantt chart also.
    //round robin algorithm.
    double time_quantum;
    cout<<"\n\t\tEnter the time slice/quantum for RR algorithm: ";
    cin>>time_quantum;
    sorting(n,process_arr);
    queue ready_queue;
    double time=process_arr[0].a_time;//time scale
    double ta_time_avg=0, w_time_avg=0;
    int arrived[n],completed[n],sequence[n];
    sequence[0]=0;
    int index,check;
    double b_time_arr[n];
    for(int i=0;i<n;++i)
```

```
{
    process_arr[i].w_time=0;
    b_time_arr[i]=process_arr[i].b_time;
    arrived[i]=0;
    completed[i]=0;
}
arrived[0]=1;
process_arr[0].s_time=time;
b_time_arr[0]-=time_quantum;
ready_queue.insert(0);
index=ready_queue.delete_();
time=time+time_quantum;
while(1)
{
    if(index==-1)
    {
        cout<<"\n\t\tAll the processes have been executed/completed.";
        break;
    }
    while(1)
    {
        check=check_arrived(n,time,process_arr,arrived);
        if(check==0)
        {
            time=time+time_quantum;
            b_time_arr[index]-=time_quantum;

        }
        else
        {
            for(int i=0;i<n;++i)
            {
                if(arrived[i]!=1&&completed[i]!=1)
                {
                    if(process_arr[i].a_time<=time)
                    {
                        arrived[i]=1;
                        ready_queue.insert(i);
                    }

                }
            }

            time=time+time_quantum;
            process_arr[index].w_time+=time-process_arr[index].a_time;
            process_arr[index].s_time=time;
            if(b_time_arr[index]==0)
            {
                completed[index]=1;
                process_arr[index].ta_time=process_arr[index].b_time+process_arr[index].w_time;
                ta_time_avg=ta_time_avg+process_arr[index].ta_time;
                w_time_avg=w_time_avg+process_arr[index].w_time;

            }
            else
            {
                ready_queue.insert(index);
            }
            break;

        }
        }
        index=ready_queue.delete_();

    }
    cout<<"\n\t\t\tAverage Waiting time is: "<<w_time_avg/n;
    cout<<"\n\t\t\tAverage Turn Around time is: "<<ta_time_avg/n;


}*/

void process_scheduling::SJF(int n, processnode process_arr[])
{
    sorting(n,process_arr);//sorting
    double time=process_arr[0].a_time+process_arr[0].b_time;
    //time scale after execution of first arrived process
    process_arr[0].s_time=process_arr[0].a_time;
    process_arr[0].w_time=0;
    process_arr[0].ta_time=process_arr[0].w_time+process_arr[0].b_time;
    double ta_time_avg=process_arr[0].ta_time, w_time_avg=0;
    int arrived[n],completed[n],sequence[n];
    sequence[0]=0;
    arrived[0]=1;
    completed[0]=1;
    for(int i=1;i<n;++i)
    {
        arrived[i]=0;
        completed[i]=0;
    }
    int index,k=1;
    while(1)
    {
        for(int i=0;i<n;++i)
        {
            if(completed[i]!=1)
            {
                if(process_arr[i].a_time<=time)
                {
                    arrived[i]=1;
                }
            }
        }
        index=shortes_b_time(n,arrived,completed,process_arr);
        if(index==-1)
        {
            //cout<<"\n\t\tNo process has arrived or all processes has been completed";
            break;
        }
        process_arr[index].s_time=time;
        process_arr[index].w_time=process_arr[index].s_time-process_arr[index].a_time;
        process_arr[index].ta_time=process_arr[index].w_time+process_arr[index].b_time;
        time=time+process_arr[index].b_time;
        completed[index]=1;
        w_time_avg=w_time_avg+process_arr[index].w_time;
        ta_time_avg=ta_time_avg+process_arr[index].ta_time;
        sequence[k]=index;
        ++k;

    }
    cout<<"******************Shortest Job First Algorithm******************";
    cout<<"\n\n\tProcess:  \t:Turnaround Time: Waiting Time\n\n";
    for(int i=0;i<n;++i)
    {
        cout<<"\tProcess: "<<process_arr[sequence[i]].p_name;
        cout<<"\t:\t"<<process_arr[sequence[i]].ta_time;
        cout<<"\t:\t"<<process_arr[sequence[i]].w_time;
        cout<<endl;
    }

    cout<<"\n\t\tGantt Representation of SJF algorithm is as follows: ";
    cout<<"\n\t\tOrder of processes is as follows: ";
    for(int i=0;i<n;++i)
    {
        if(i==n-1)
        {

            break;
        }
        cout<<process_arr[sequence[i]].p_name<<"-->";
    }
    cout<<"\n\t\t"<<process_arr[0].s_time;
    for(int i=0;i<n;++i)
    {
        for(int j=0;j<int(process_arr[sequence[i]].b_time);++j)
        {
            cout<<" _";
        }
        if(i==n-1)
        {
            cout<<process_arr[sequence[i]].s_time+process_arr[sequence[i]].b_time;
        }
        else
        cout<<process_arr[sequence[i+1]].s_time;
    }
    cout<<"\n\t\t\tAverage Waiting time is: "<<w_time_avg/n;
    cout<<"\n\t\t\tAverage Turn Around time is: "<<ta_time_avg/n;
}

void process_scheduling::FCFS(int n, processnode process_arr[])
{

    sorting(n,process_arr);
    process_arr[0].s_time=process_arr[0].a_time;
    process_arr[0].w_time=0;
    process_arr[0].ta_time=process_arr[0].b_time;
    double ta_time_avg=process_arr[0].ta_time, w_time_avg=0;
    for(int i=1;i<n;++i)
    {
        process_arr[i].s_time=process_arr[i-1].s_time+process_arr[i-1].b_time;
        process_arr[i].w_time=process_arr[i].s_time-process_arr[i].a_time;
        if(process_arr[i].w_time<0)
        {
            //negative waiting time means process arrives later and did'nt have to wait
            process_arr[i].w_time=0; //waiting time becomes 0
        }
        process_arr[i].ta_time=process_arr[i].w_time+process_arr[i].b_time;
        w_time_avg=w_time_avg+process_arr[i].w_time;
        ta_time_avg=ta_time_avg+process_arr[i].ta_time;
    }
    cout<<"*******************First Come First Served Algorithm*******************";
    cout<<"\n\n\tProcess:  \t:Turnaround Time: Waiting Time\n\n";
    for(int i=0;i<n;++i)
    {
        cout<<"\tProcess: "<<process_arr[i].p_name<<"\t:\t"<<process_arr[i].ta_time;
        cout<<"\t:\t"<<process_arr[i].w_time;
        cout<<endl;
    }
    cout<<"\n\t\tGantt Representation of FCFS algorithm is as follows: ";
    cout<<"\n\t\tOrder of processes is as follows: ";
    for(int i=0;i<n;++i)
    {
        if(i==n-1)
        {
            cout<<process_arr[i].p_name;
            break;
        }
        cout<<process_arr[i].p_name<<"-->";
    }
    cout<<"\n\t\t"<<process_arr[0].s_time;
    for(int i=0;i<n;++i)
    {
        for(int j=0;j<int(process_arr[i].b_time);++j)
        {
            cout<<" _";
        }
        if(i==n-1)
        {
            cout<<process_arr[i].s_time+process_arr[i].b_time;
        }
        else
        cout<<process_arr[i+1].s_time;
    }
    cout<<"\n\t\t\tAverage Waiting time is: "<<w_time_avg/n;
    cout<<"\n\t\t\tAverage Turn Around time is: "<<ta_time_avg/n;



}

void process_scheduling::input_processes()
{
    int n,choice;
    cout<<"\n\t\t\tOS CPU Process Scheduling Assignment";
    cout<<"\n\t\t\t\tBy Sanskar Sharma";
    cout<<"\n\t\t\t\t PRN 0120180381";
    cout<<"\n\tEnter the number of processes: ";
    cin>>n;
    processnode process_arr[n];
    for(int i=0;i<n;++i)
    {
        cout<<"\n\t\tEnter the process "<<i+1<<" name: ";
        cin>>process_arr[i].p_name;
        cout<<"\n\t\tEnter the arrival time of process "<<i+1<<" : ";
        cin>>process_arr[i].a_time;
        if(process_arr[i].a_time<0)
        {
            process_arr[i].a_time=0;
        }
        cout<<"\n\t\tEnter the burst time of process "<<i+1<<" : ";
        cin>>process_arr[i].b_time;
        while(process_arr[i].b_time<=0)
        {
            cout<<"\n\t\t\tEnter a positive busrt time!! Enter again: ";
            cin>>process_arr[i].b_time;
        }

    }
    while(1)
    {
        cout<<"\n\tChoose an Algorithm to schedule the processes: ";
        cout<<"\n\t\tNon-Preemptive Algorithms: \n\t\t  ";
        cout<<"1. First Come First Served (FCFS).";
        cout<<"\n\t\t  2. Shortest Job First (SJF).";
        cout<<"\n\t\tPreemptive Algorithms: \n\t\t  ";
        cout<<"3. Round Robin Algorithm.";
        cout<<"\n\t4. Exit";
        cout<<"\n\tEnter your choice: ";
        cin>>choice;
        switch(choice)
        {
            case 1:
                FCFS(n,process_arr);
                break;
            case 2:
                SJF(n,process_arr);
                break;
            case 3:
                Roundrobin(n,process_arr);
                break;
            case 4:
                exit(0);//Sucessfully exited
            default:
                //unsuccessful exit.
                exit(-1);
        }
    }

}

int main()
{
    process_scheduling ps;
    ps.input_processes();

    return 0;

}


/*
Output:
Problem 1: Solved by FCFS and SJF respectively.


                        OS CPU Process Scheduling Assignment
                                By Sanskar Sharma
                                 PRN 0120180381
        Enter the number of processes: 4

                Enter the process 1 name: Task1

                Enter the arrival time of process 1 : 2

                Enter the burst time of process 1 : 3
                Enter the process 2 name: Task2

                Enter the arrival time of process 2 : 4

                Enter the burst time of process 2 : 2

                Enter the process 3 name: Task3

                Enter the arrival time of process 3 : 5

                Enter the burst time of process 3 : 1

                Enter the process 4 name: Task4

                Enter the arrival time of process 4 : 7

                Enter the burst time of process 4 : 4

        Choose an Algorithm to schedule the processes:
                Non-Preemptive Algorithms:
                  1. First Come First Served (FCFS).
                  2. Shortest Job First (SJF).
                Preemptive Algorithms:
                  3. Round Robin Algorithm.
        4. Exit
        Enter your choice: 1
******************First Come First Served Algorithm******************

        Process:        :Turnaround Time: Waiting Time

        Process: Task1  :       3       :       0
        Process: Task2  :       3       :       1
        Process: Task3  :       3       :       2
        Process: Task4  :       5       :       1

                Gantt Representation of FCFS algorithm is as follows:
                Order of processes is as follows: Task1-->Task2-->Task3-->Task4
                2 _ _ _5 _ _7 _8 _ _ _ _12
                        Average Waiting time is: 1
                        Average Turn Around time is: 3.5
        Choose an Algorithm to schedule the processes:
                Non-Preemptive Algorithms:
                  1. First Come First Served (FCFS).
                  2. Shortest Job First (SJF).
                Preemptive Algorithms:
                  3. Round Robin Algorithm.
        4. Exit
        Enter your choice: 2
******************Shortest Job First Algorithm******************

        Process:        :Turnaround Time: Waiting Time

        Process: Task1  :       3       :       0
        Process: Task3  :       1       :       0
        Process: Task2  :       4       :       2
        Process: Task4  :       5       :       1

                Gantt Representation of SJF algorithm is as follows:
                Order of processes is as follows: Task1-->Task3-->Task2-->Task4
                2 _ _ _5 _6 _ _8 _ _ _ _12
```

```
                        Average Waiting time is: 0.75
                        Average Turn Around time is: 3.25
        Choose an Algorithm to schedule the processes:
                Non-Preemptive Algorithms:
                  1. First Come First Served (FCFS).
                  2. Shortest Job First (SJF).
                Preemptive Algorithms:
                  3. Round Robin Algorithm.
        4. Exit
        Enter your choice: 4

--------------------------------
Process exited after 87.25 seconds with return value 0
Press any key to continue . . .
```

Problem 2: Solved by Round Robin

```
                        OS CPU Process Scheduling Assignment
                                By Sanskar Sharma
                                 PRN 0120180381
        Enter the number of processes: 4

                Enter the process 1 name: Task1

                Enter the arrival time of process 1 : 1

                Enter the burst time of process 1 : 4

                Enter the process 2 name: Task2

                Enter the arrival time of process 2 : 2

                Enter the burst time of process 2 : 3

                Enter the process 3 name: Task3

                Enter the arrival time of process 3 : 3

                Enter the burst time of process 3 : 5

                Enter the process 4 name: Task4

                Enter the arrival time of process 4 : 4

                Enter the burst time of process 4 : 7

        Choose an Algorithm to schedule the processes:
                Non-Preemptive Algorithms:
                  1. First Come First Served (FCFS).
                  2. Shortest Job First (SJF).
                Preemptive Algorithms:
                  3. Round Robin Algorithm.
        4. Exit
        Enter your choice: 3

                Enter the time slice/quantum for RR algorithm: 2
*******************Round Robin Algorithm*******************

        Process:        :Turnaround Time: Waiting Time
        Process: Task1  :       9       :       5
        Process: Task2  :       9       :       6
        Process: Task3  :       13      :       8
        Process: Task4  :       15      :       8

                        Average Waiting time is: 6.75
                        Average Turn Around time is: 11.5
        Choose an Algorithm to schedule the processes:
                Non-Preemptive Algorithms:
                  1. First Come First Served (FCFS).
                  2. Shortest Job First (SJF).
                Preemptive Algorithms:
                  3. Round Robin Algorithm.
        4. Exit
        Enter your choice: 4

--------------------------------
Process exited after 48.91 seconds with return value 0
Press any key to continue . . .

*/
```