Name: Anas Ahmad Ilyas Ahmad Batch:1

Roll No: 222010001

**Source Code:**

#include <bits/stdc++.h>

using namespace std;

class Process{

public:

int p\_id; //process id

int at; //arrival time

int bt; //burst time

int rbt; //remaining burst time

int st; //start time

int ct; //completion time

int tat; //turnaround time

int wt; //waiting time

int rt; //response time

};

bool findVal(deque<int> q,int val){

deque<int>::iterator itr;

itr = find(q.begin(), q.end(), val);

if(itr != q.end())

return false;

else

return true;

}

int main(){

//Anas Ahmad Ilyas Ahmad Batch No - 1 Roll No - 1

int n, tq;

cout << "Enter the number of processes:";

cin >> n;

cout << "Enter the time quantum:";

cin >> tq;

Process p[n];

bool is\_completed[n] = {false};

deque<int> running\_queue;

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

cout << "Enter arrival time and burst time of process " << i + 1 << ": ";

cin >> p[i].at >> p[i].bt;

p[i].rbt = p[i].bt;

p[i].p\_id = i + 1;

}

int completed = 0;

int current\_time = 0;

int total\_tat = 0;

int total\_wt = 0;

int total\_rt = 0;

int curr\_process = -1;

while (completed != n){

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

if (curr\_process == i)

continue;

if (p[i].at <= current\_time && is\_completed[i] == false)

if(findVal(running\_queue,i))

running\_queue.push\_back(i);

}

if(curr\_process != -1 && is\_completed[curr\_process]==false)

running\_queue.push\_back(curr\_process);

if(!running\_queue.empty())

curr\_process = running\_queue.front();

else

curr\_process = -1;

if (curr\_process != -1){

if (p[curr\_process].rbt == p[curr\_process].bt || p[curr\_process].rbt <= tq){

if(p[curr\_process].rbt == p[curr\_process].bt)

p[curr\_process].st = current\_time;

if (p[curr\_process].rbt <= tq){

current\_time = current\_time + p[curr\_process].rbt;

p[curr\_process].ct = current\_time;

p[curr\_process].tat = p[curr\_process].ct - p[curr\_process].at;

p[curr\_process].wt = p[curr\_process].tat - p[curr\_process].bt;

p[curr\_process].rt = p[curr\_process].st - p[curr\_process].at;

total\_tat = total\_tat + p[curr\_process].tat;

total\_wt = total\_wt + p[curr\_process].wt;

total\_rt = total\_rt + p[curr\_process].rt;

running\_queue.pop\_front();

is\_completed[curr\_process] = true;

completed++;

}

else

{

p[curr\_process].rbt = p[curr\_process].rbt - tq;

current\_time = current\_time + tq;

running\_queue.pop\_front();

}

}

else

{

p[curr\_process].rbt -= tq;

current\_time += tq;

running\_queue.pop\_front();

}

}

else

{

current\_time++;

}

}

cout<<endl;

cout<<"------Round Robin CPU Scheduling Algorithm------\n"<<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].p\_id << "\t" << p[i].at << "\t" << p[i].bt << "\t" << p[i].st << "\t" << p[i].ct << "\t" << p[i].tat << "\t" << p[i].wt << "\t" << p[i].rt << "\t"<< "\n"<< endl;

}

cout << "Average Turnaround Time: " << (float)total\_tat / n << endl;

cout << "Average Waiting Time: " << (float)total\_wt / n << endl;

cout << "Average Response Time: " << (float)total\_rt / n << endl;

return 0;

}

**Output:**

