**Description:**Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time, he wishes to dedicate a fixed amount of time (quantum) to every request so that everyone gets a fair share of his time. He will log into the system from 10am to 12am only. He wants to have separate requests queues for students and faculty. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time.

Round Robin scheduling algorithm is one of the most popular scheduling algorithms which can actually be implemented in most of the operating systems. This is the **preemptive version** of first come first serve scheduling. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a **cyclic way**. A certain time slice is defined in the system which is called time **quantum**. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **terminate** else the process will go back to the **ready queue** and waits for the next turn to complete the execution.

1. The algorithm for proposed solution of the assigned problem.

**Algorithm:**

**RoundRobin Algorithm**

1. **Take the details of all the processes (No. of processes, BT, AT, Quantum) as input from the user using getSystem( ) function.**
2. **Call the schedule( ) function to determine the schedule/sequence of operations.**
3. **Implement addArrivedprocesstoqueue( ) and addtoqueue( ) methods to arrange the operations in ascending order of Arrival times.**
4. **Calculate the total waiting time and average waiting time using WaitingTime( ) function.**
5. **Print the result using printSystem( ) function.**
6. **Exit**

**The implementation of this algorithm using C is given in the next page.**

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

int processes[100][3], NP, quantum, scheduler[1000],WT[100];

unsigned int time = 0;

typedef struct el

{

unsigned int p;

struct el \* next;

}Q;

Q \* qeue = NULL;

void getSystem()

{

int i;

printf("\nNumber of processes: ");

scanf("%d", &NP);

printf("\nThe Quantum: ");

scanf("%d", &quantum);

for(i=0; i<NP; i++ )

{

printf("\n Arrival Time of p%d: ", i);

scanf("%d", &processes[i][0]);

printf("\n Burst time for p%d: ", i);

scanf("%d", &processes[i][1]);

processes[i][2] = processes[i][1];

printf("\n-----------");

}

}

void printSystem()

{

int i;

printf("\n\t\tOur System is :");

printf("\nQuantum: %d",quantum);

printf("\nPi: AT BT RT");

for(i=0; i<NP; i++)

{

printf("\nP%d: %d %d %d", i, processes[i][0], processes[i][1], processes[i][2]);

}

printf("\nThe queue: ");

Q \*n;

for(n=qeue; n!=NULL; n=n->next)

{

printf("P%d ",n->p);

}

}

unsigned int executionRemained()

{

int i;

unsigned int x = 0;

for(i=0; i<NP; i++)

{

if(processes[i][2] > 0)

{

x = 1;

}

}

return x;

}

void addToQeue(int i)

{

Q \*n, \*n1;

n = (Q \*)malloc(sizeof(Q));

n->next = NULL;

n->p = i;

if(qeue == NULL)

{

qeue = n;

}

else

{

for(n1 = qeue ; n1->next!=NULL; n1=n1->next);

n1 -> next = n;

}

}

void addArrivedProcessesToQeue()

{

int i;

for(i=0; i<NP; i++)

{

if(processes[i][0] == time)

{

addToQeue(i);

}

}

}

unsigned int getNextProcess()

{

Q \*n;

int x;

if(qeue == NULL)

{

return -1;

}

else

{

x = qeue -> p;

n = qeue;

qeue = qeue -> next;

free(n);

return x;

}

}

void schedule()

{

unsigned int np, toRun, q, i;

q = 0;

addArrivedProcessesToQeue();

while(executionRemained())

{

np = getNextProcess();

if(np == -1)

{

scheduler[time] = -1;

time++;

addArrivedProcessesToQeue();

}

else

{

q = quantum;

if(processes[np][2] < q)

{

q = processes[np][2];

}

for(i = q; i>0; i--)

{

scheduler[time]=np;

time++;

processes[np][2]--;

addArrivedProcessesToQeue();

}

if(processes[np][2] > 0)

{

addToQeue(np);

}

}

printSystem();

int x;

}

}

void printScheduling()

{

int i;

printf("\n\nScheduling: \n");

for(i=0; i<time; i++)

{

printf("[%d-%d] (P%d) \n",i,i+1 ,scheduler[i]);

}

printf("\n\nWaiting Time: \n");

for(i=0; i<NP; i++)

{

printf("\nP%d: %d", i, WT[i]);

}

//counting Average Waiting Time...

float AWT = 0.0;

for(i=0; i<NP; i++)

{

AWT = AWT+WT[i];

}

printf("\n\nTotal Waiting Time: %f", AWT);

AWT = AWT/NP;

printf("\n\nAverage Waiting Time: %f", AWT);

}

void WatingTime()

{

int i;

unsigned int releaseTime, t;

for(i=0; i<NP; i++)

{

for(t=time-1; scheduler[t]!= i; t--);

releaseTime = t+1;

WT[i] = releaseTime - processes[i][0] - processes[i][1];

}

}

main()

{

int ch;

printf("Which queue would you like to handle?");

printf("\n1.Student Queue\n2.Faculty Queue\n");

printf("Enter your choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1:

getSystem();

printSystem();

schedule();

WatingTime();

printScheduling();

break;

case 2:

getSystem();

printSystem();

schedule();

WatingTime();

printScheduling();

break;

default:

printf("Invalid choice entered");

break;

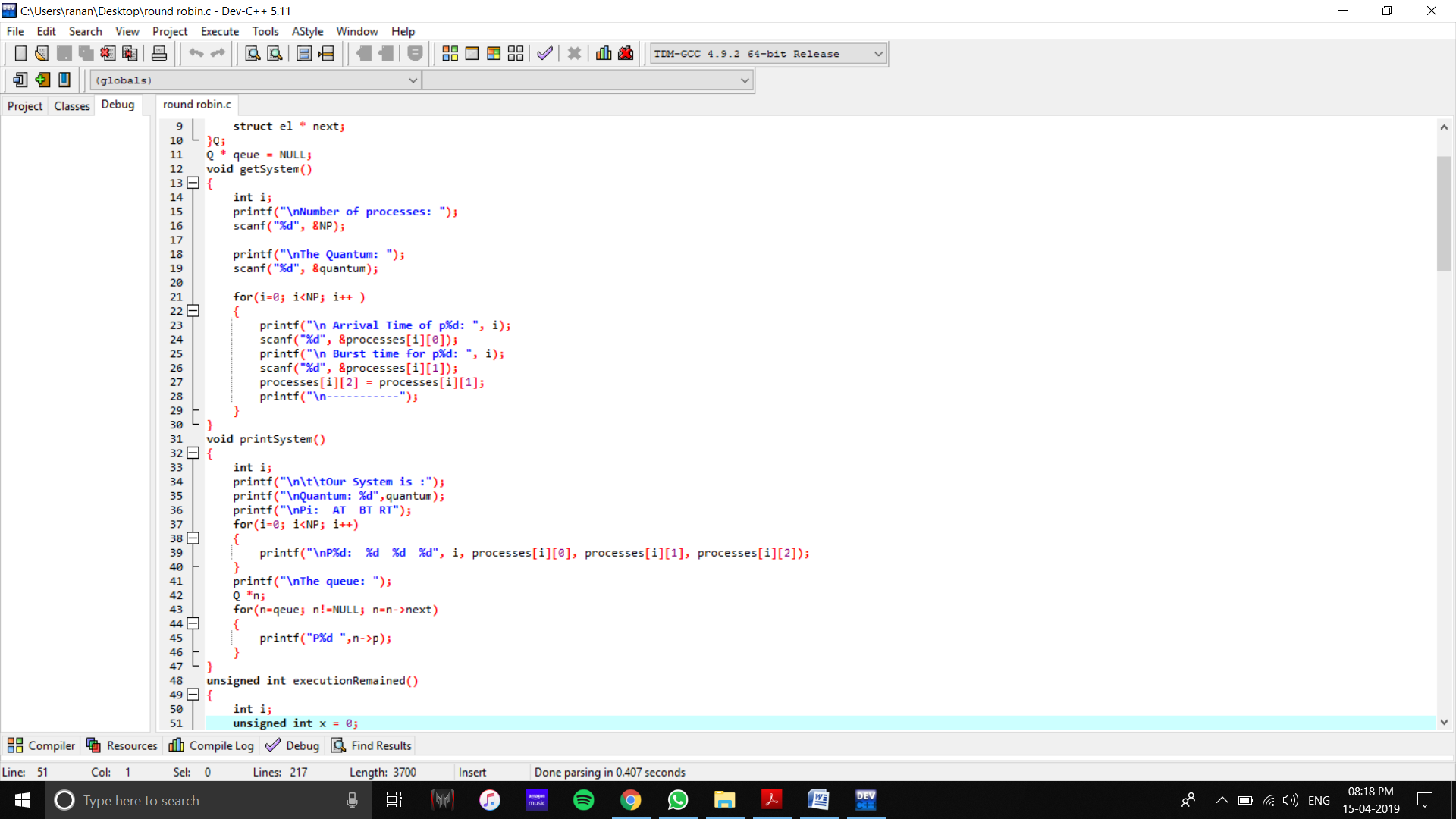
}

getch();

}

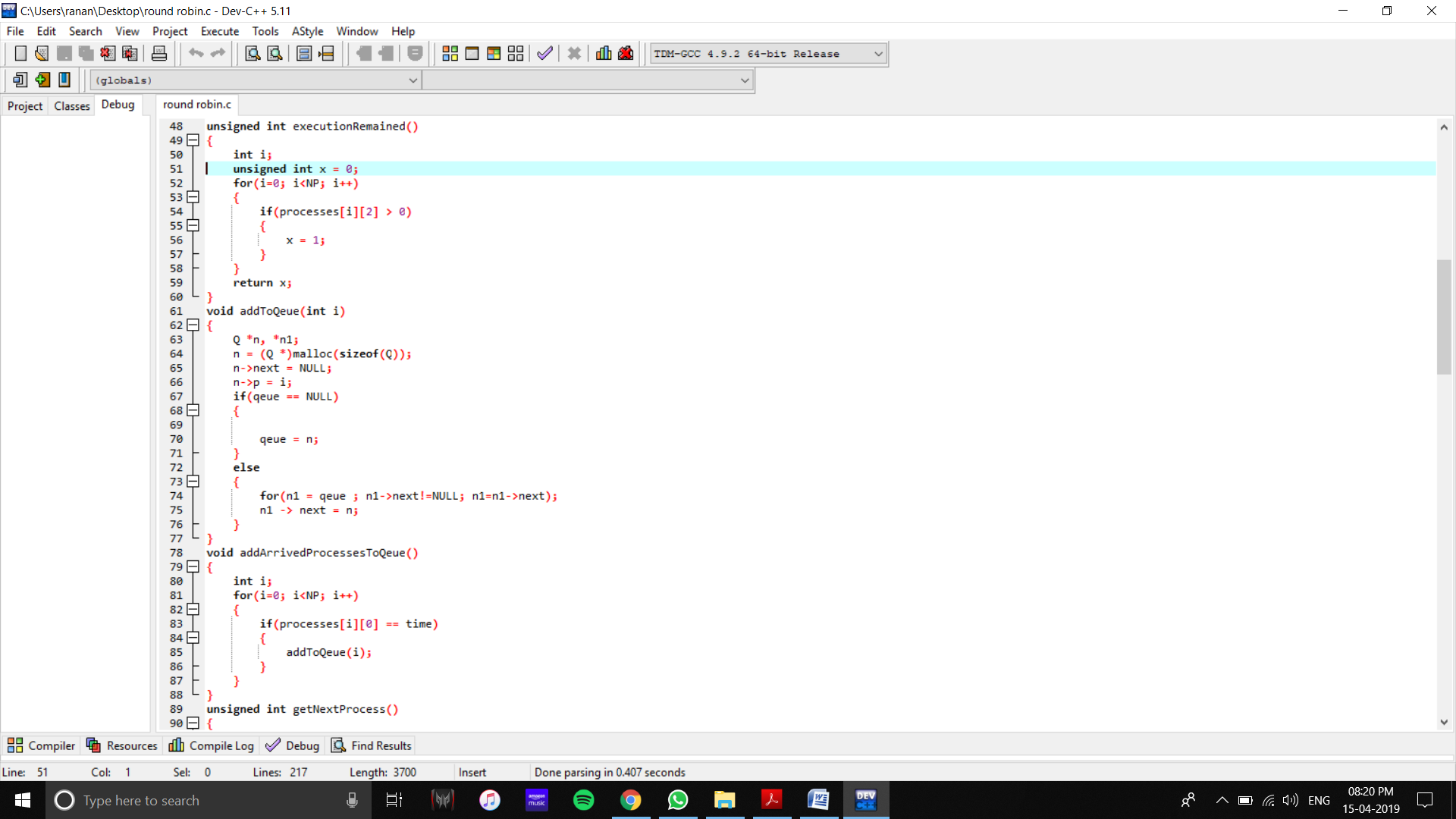
1. Calculate complexity of implemented algorithm. (Student must specify complexity of each line of code along with overall complexity)

**Description (purpose of use):**

****

Complexity – O(n)

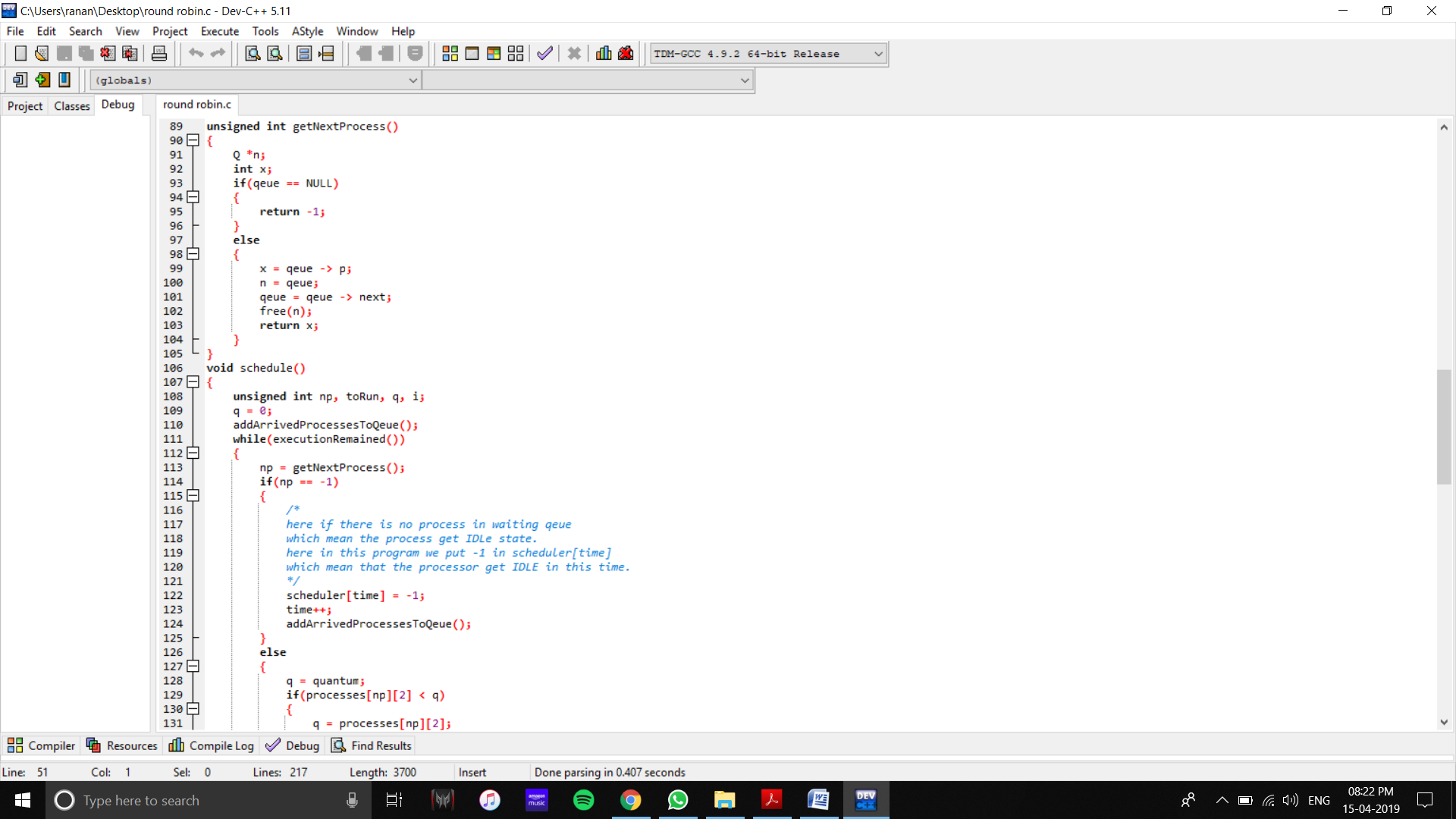
Complexity – O(n)

****

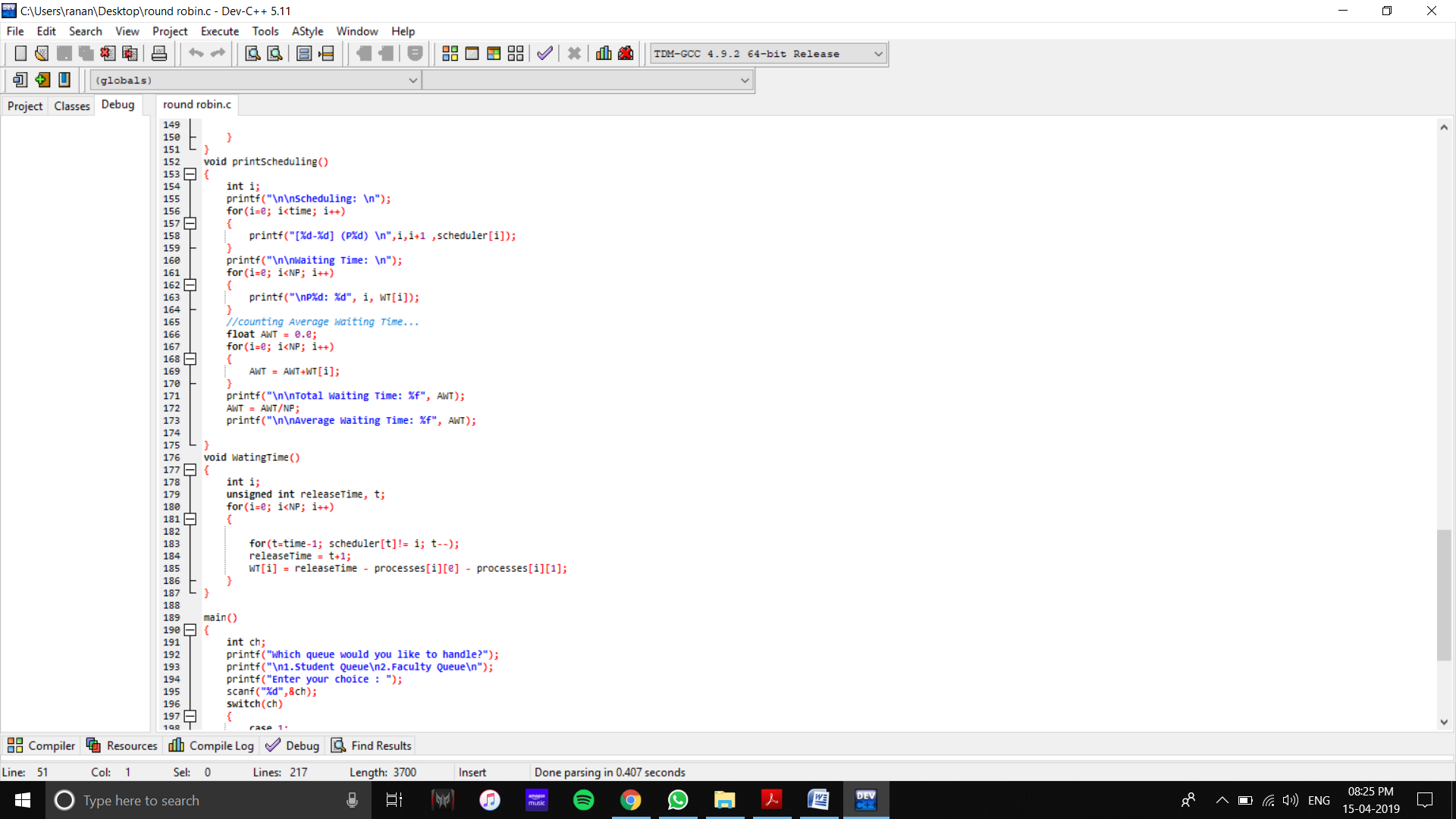
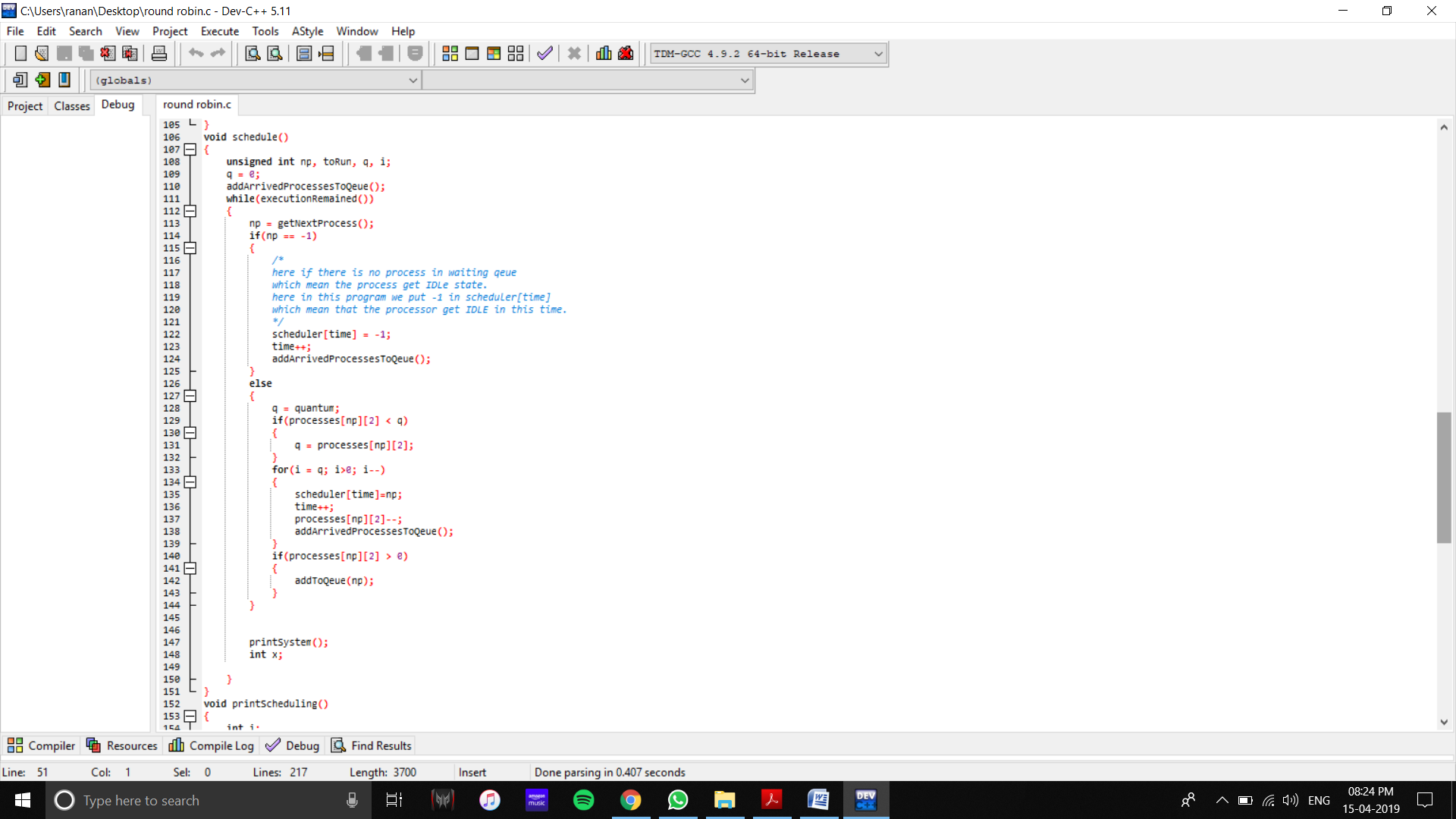
Complexity –O( n)

Complexity – O(1)

Complexity –O( n)

****

Complexity – O(n)

****

Complexity – O(n\*n\*m)

Complexity –O(n\*n)

Complexity – O(n)

**Total complexity of the program :**

**O(mn3)**

1. Explain all the constraints given in the problem. Attach the code snippet of the implemented constraint.

Code snippet:

**Constraint 1 – To determine the Schedule**

void schedule()

{

unsigned int np, toRun, q, i;

q = 0;

addArrivedProcessesToQeue();

while(executionRemained())

{

np = getNextProcess();

if(np == -1)

{

scheduler[time] = -1;

time++;

addArrivedProcessesToQeue();

}

else

{

q = quantum;

if(processes[np][2] < q)

{

q = processes[np][2];

}

for(i = q; i>0; i--)

{

scheduler[time]=np;

time++;

processes[np][2]--;

addArrivedProcessesToQeue();

}

if(processes[np][2] > 0)

{

addToQeue(np);

}

}

printSystem();

int x;

}

}

**Total waiting time and Average waiting time**

void WatingTime()

{

int i;

unsigned int releaseTime, t;

for(i=0; i<NP; i++)

{

for(t=time-1; scheduler[t]!= i; t--);

releaseTime = t+1;

WT[i] = releaseTime - processes[i][0] - processes[i][1];

}

}

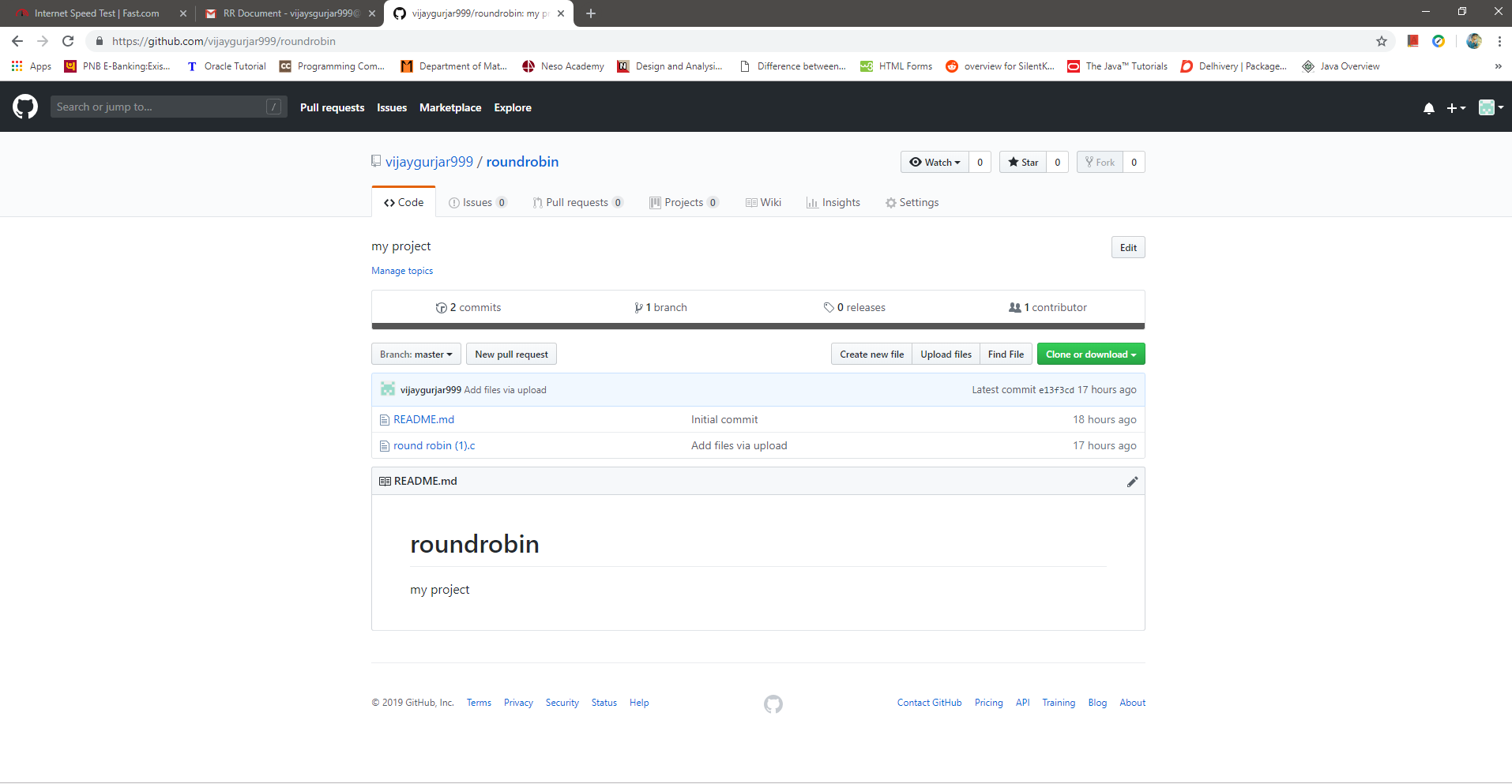
1. If you have implemented any additional algorithm to support the solution, explain the need and usage of the same.

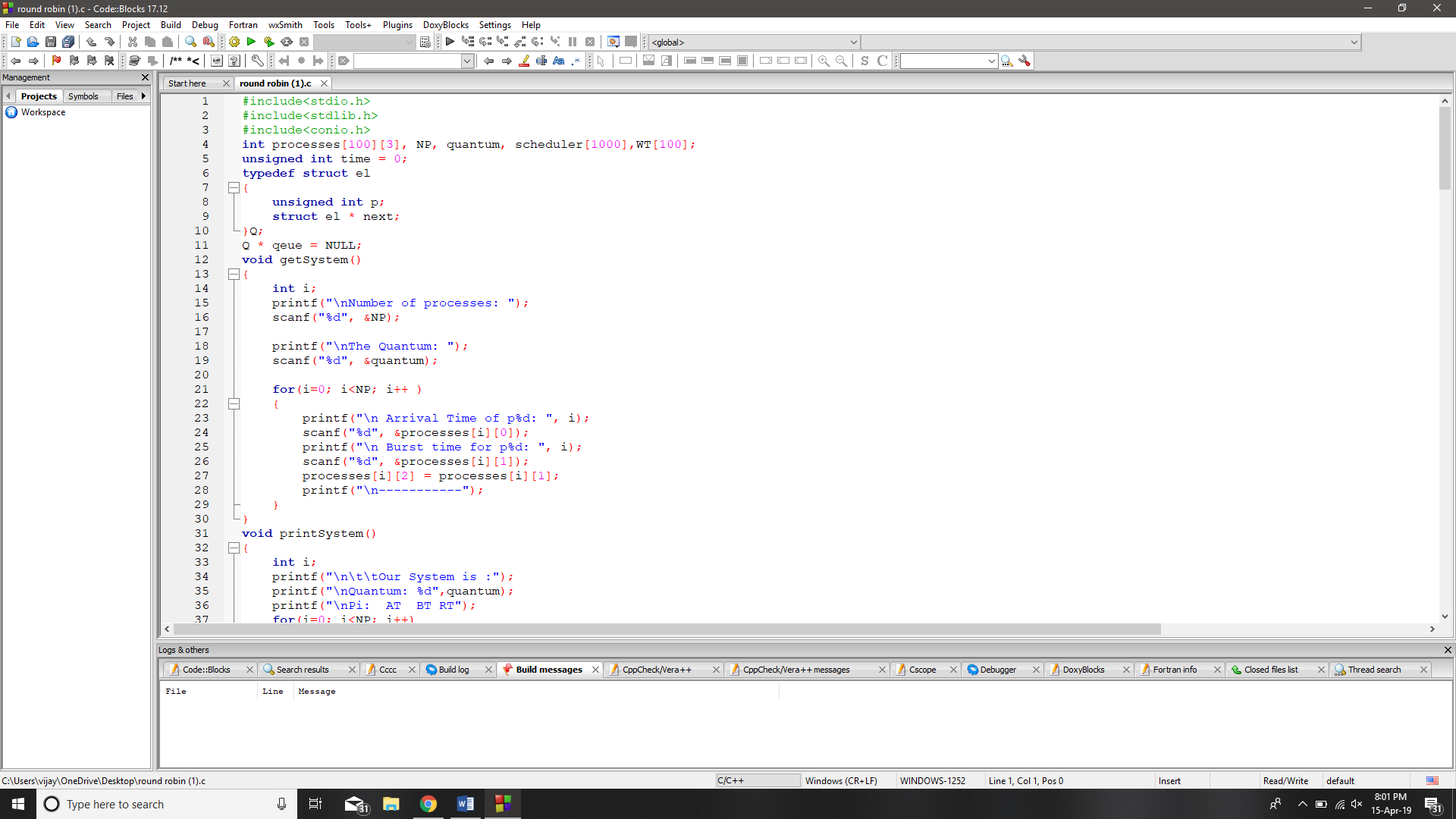
**Description: No additional algorithm implemented**

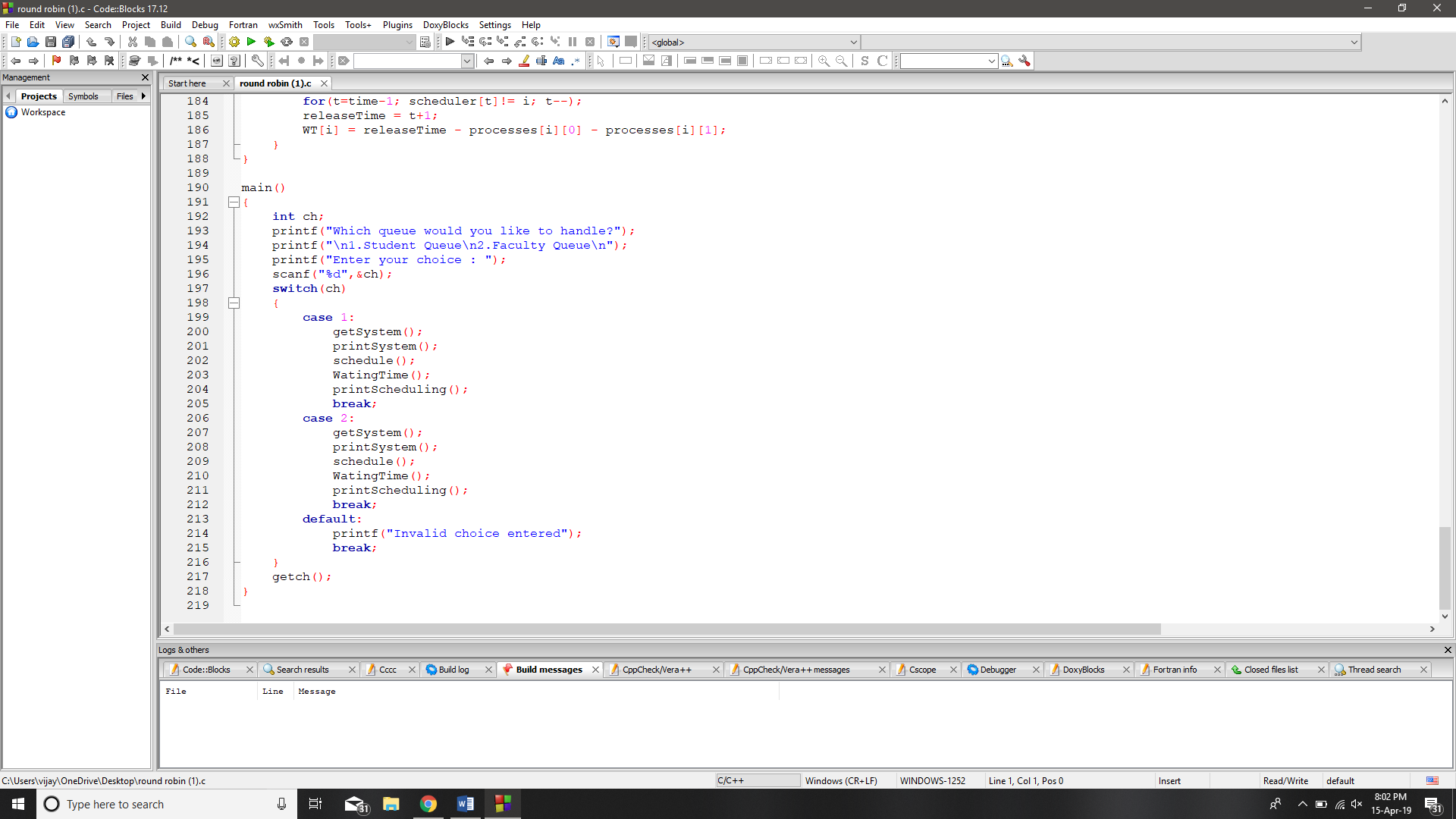
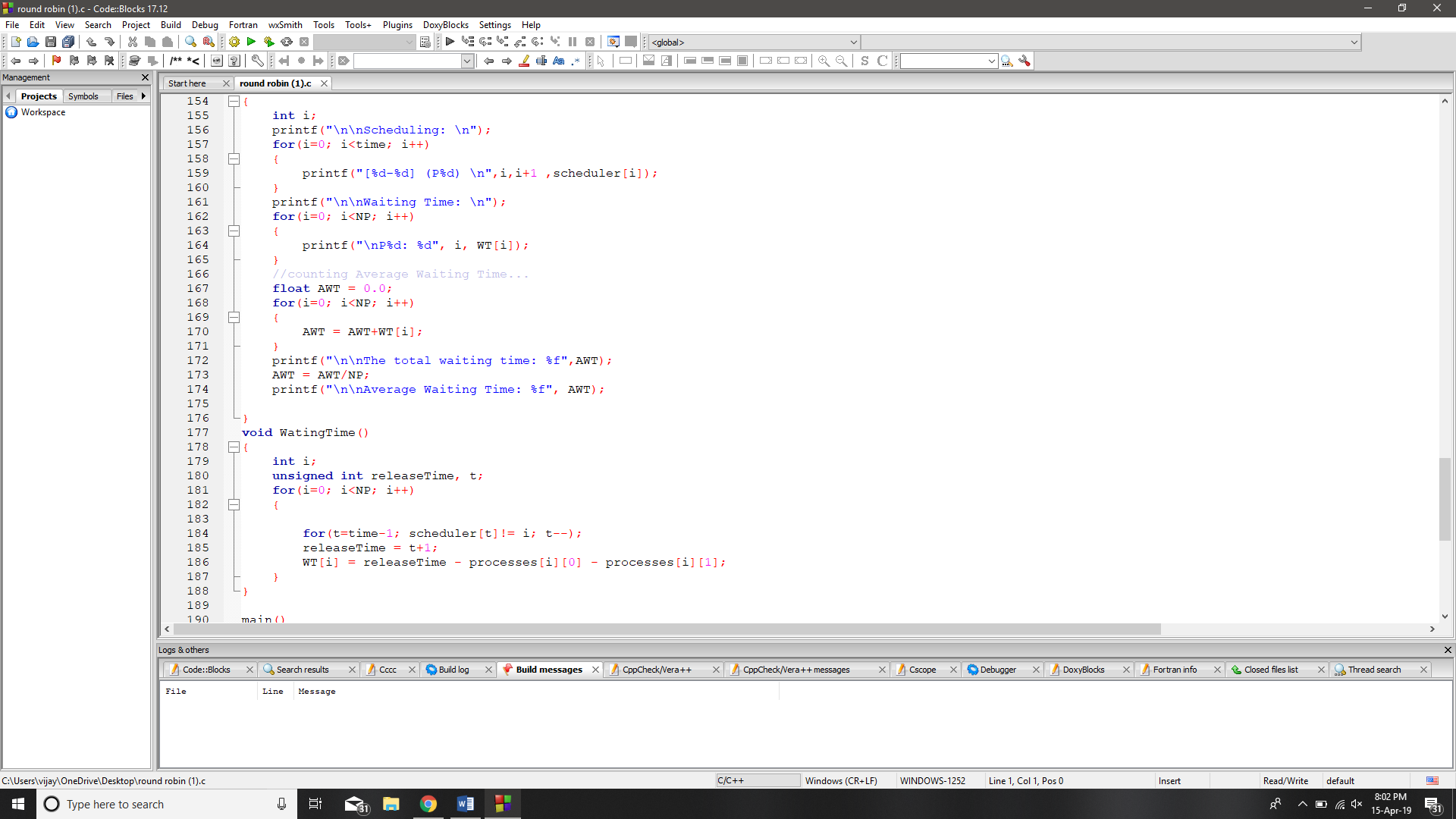
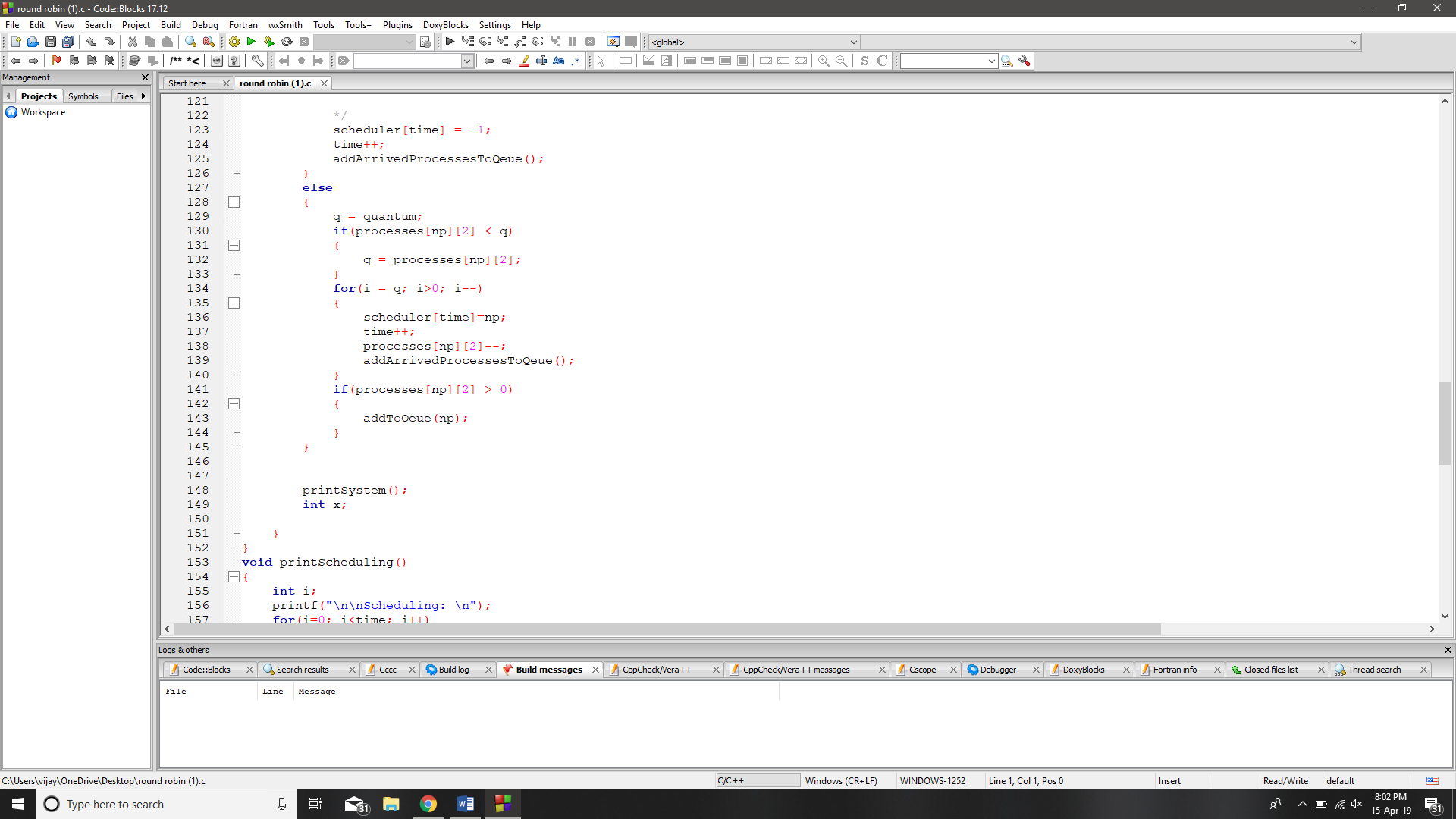
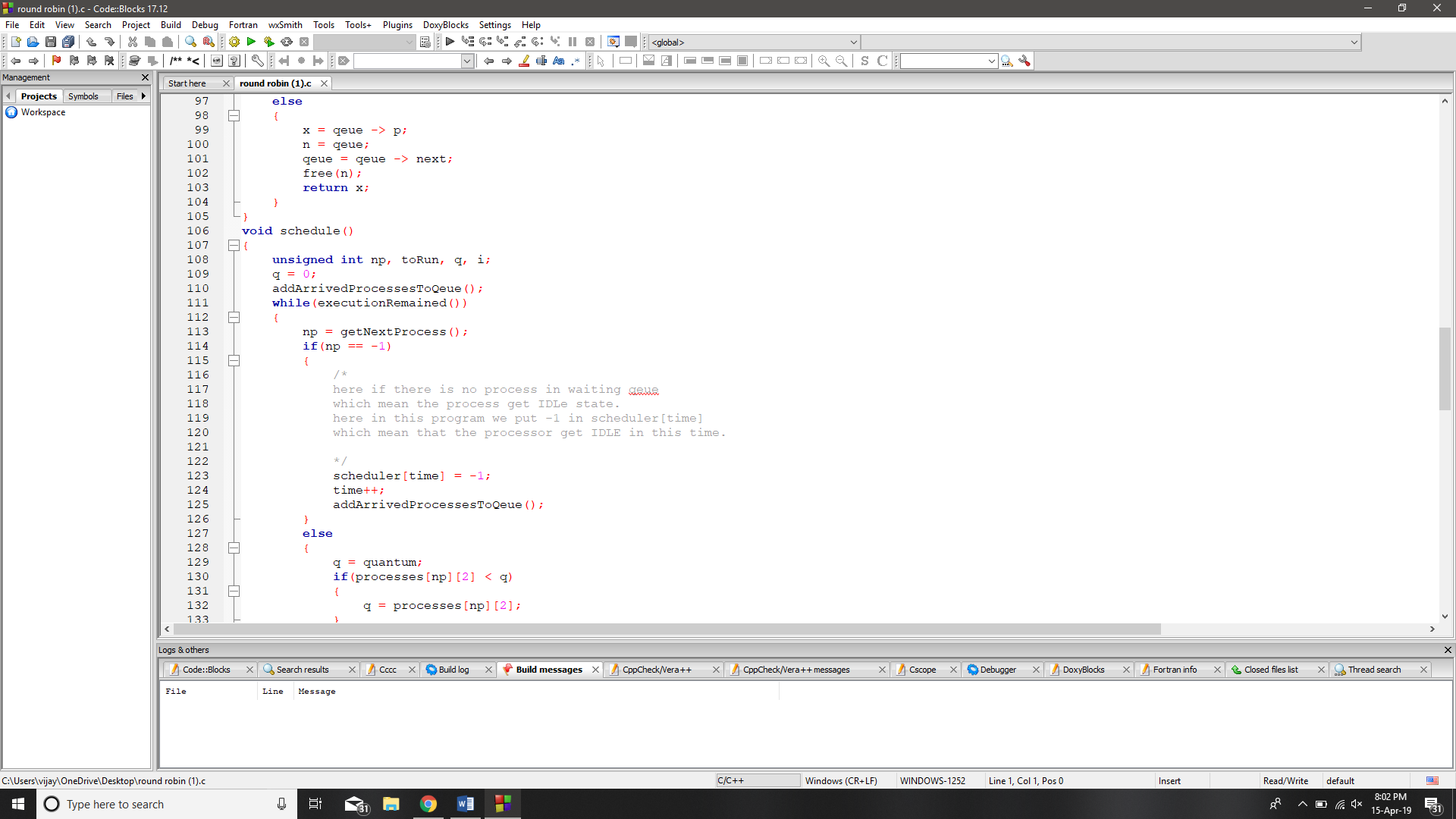
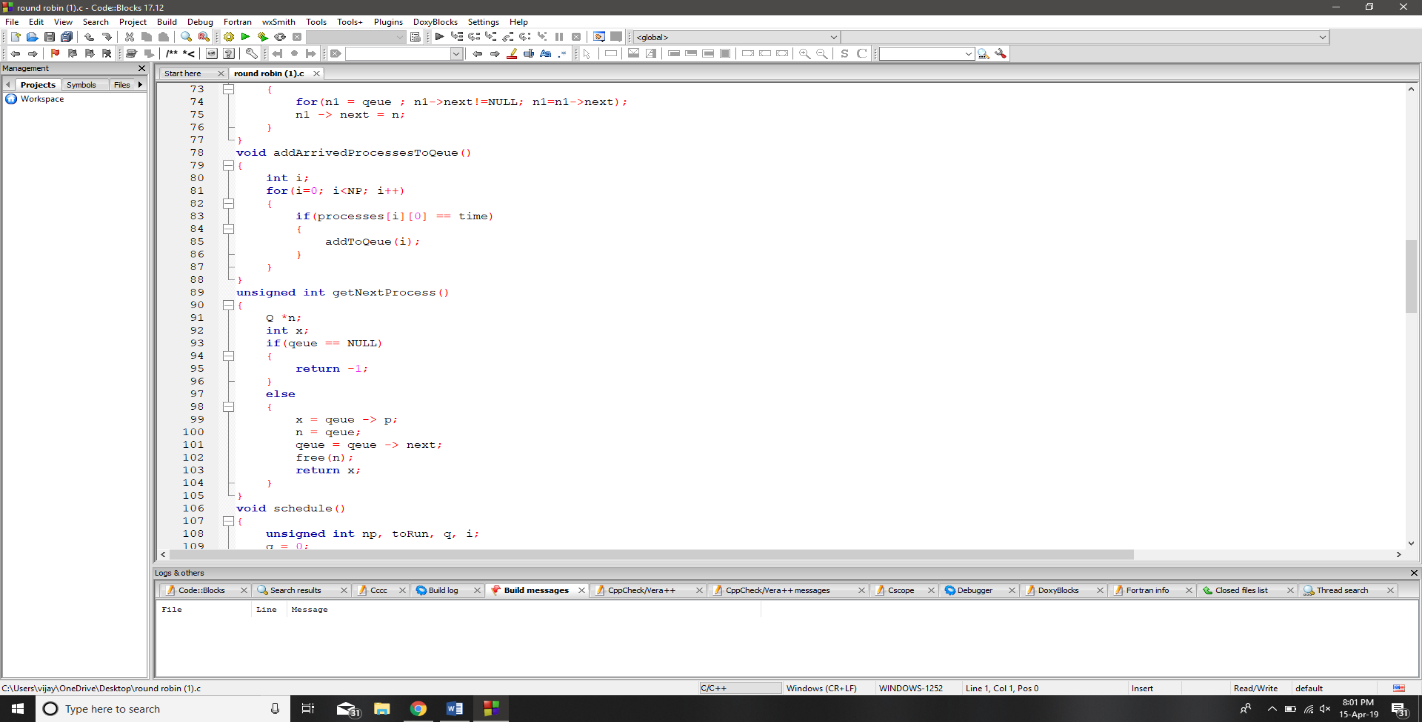
1. Explain the boundary conditions of the implemented code.

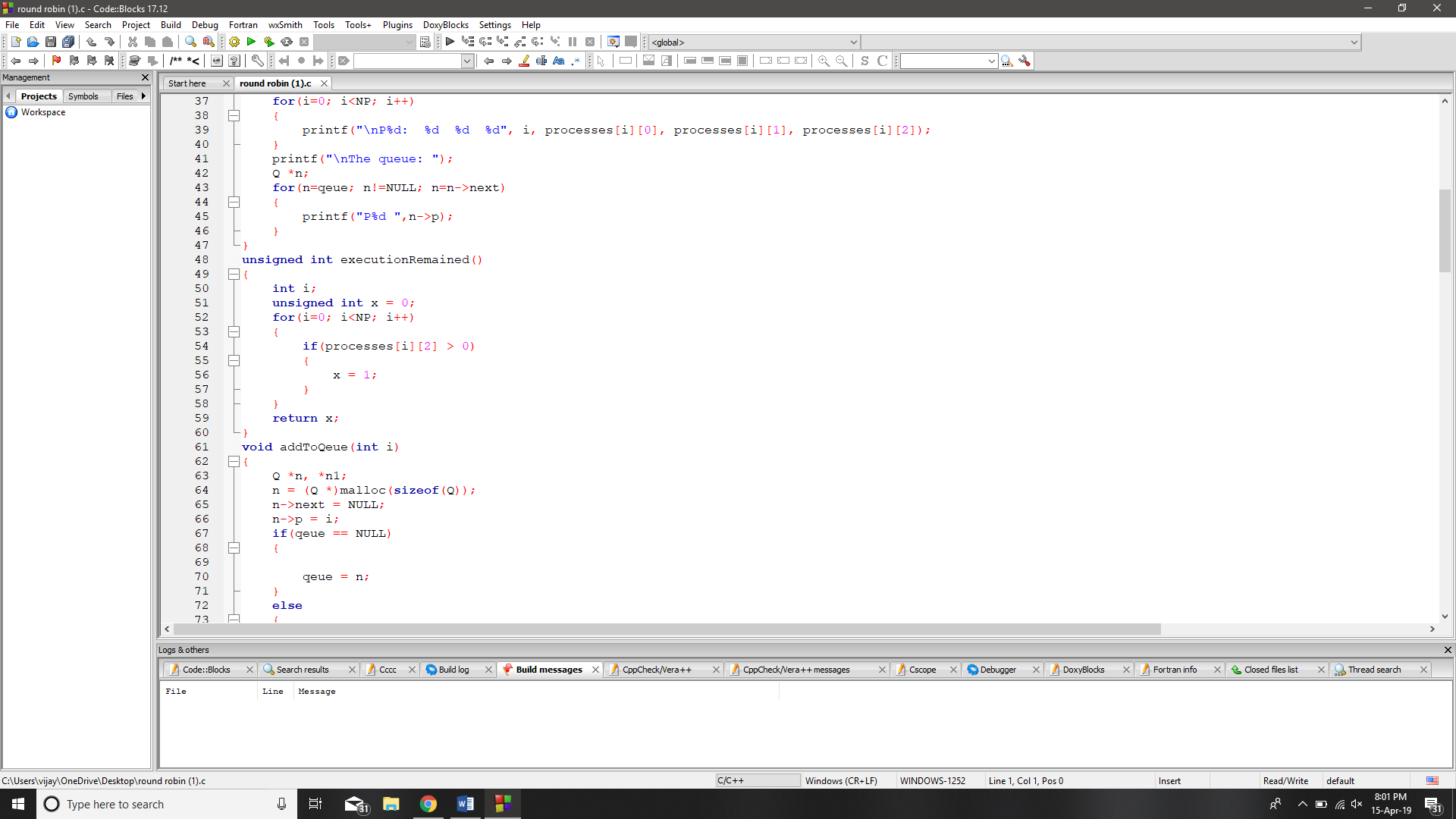
**Description:**

* If the quantum is too high, the algorithm works as FCFS.
* Waiting time has to be calculated with precision, as any error would cause a critical change in the final answer.
* The higher the time quantum, the higher the response time in the system.
* The lower the time quantum, the higher the context switching overhead in the system.
* Deciding a perfect time quantum is really a very difficult task in the system.

1. **Some Screenshoot of my code and Github profile**







1. Test cases applied on the solution of assigned problem**.**

**Description:**

**Test case 1:**

|  |  |  |
| --- | --- | --- |
| **Process** | **BT** | **AT** |
| **1** | **3** | **0** |
| **2** | **4** | **0** |
| **3** | **3** | **0** |

**Quantum= 1**

**Gantt Chart :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **1** | **2** | **3** | **1** | **2** | **3** | **2** |

1. **10**

**Waiting time:**

**P1-4**

**P2-6**

**P3-6**

**AWT=5.33**

**Test Case 2:**

|  |  |  |
| --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** |
| **1** | **0** | **5** |
| **2** | **1** | **6** |
| **3** | **2** | **3** |
| **4** | **3** | **1** |
| **5** | **4** | **5** |
| **6** | **6** | **4** |

**Quantum= 4**

**Gantt Chart :**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **5** | **1** | **6** | **2** | **5** |

**0 24**

**Waiting time:**

**P1-12**

**P2-16**

**P3-6**

**P4-8**

**P5-15**

**P6-11**

**AWT=12.66667**

1. **Have you made minimum 5 revisions of solution on GitHub?**

**I have done only 2 Rivision.**

**GitHub Link:**<https://github.com/vijaygurjar999/roundrobin>