Overview
In this project, you will implement a simulation of the round-robin scheduling algorithm and then run it on a list of processes. The input to the program will be the name of a file containing the list of processes to be scheduled, the length of time a process is unable to run when it blocks, and the quantum. The output will show the scheduling of the processes. For this assignment, you may work individually or in a group with one other student.

Setup
In CodeLite on your virtual machine:

- Create a new C++ project named Project5.
- Add two classes to the project, one named Process and one named RR_Scheduler.

Submitting Your Assignment
Submit the source code (.cpp and .h files) from your project in a zip file. Name the zip file YourLastNameYourFirstNameProject5.zip. Upload the zip file to Canvas before the due date.

Important submission notes for students working in a group:

If you worked with another student, name the zip file using the last names of both students: FirstPersonsLastNameSecondPersonsLastNameProject5.zip

One person should upload the zip files to Canvas. The other person should upload a single document stating who their team member was.

Remember that late assignments are not accepted in this course. See the syllabus for details.

Command Line
Your program should accept three arguments on the command line.

1. The input file name
2. block_duration: the decimal integer length that a process is unavailable to run after it blocks
3. quantum: the decimal integer length of the time slice for the round-robin scheduler

An error message should be printed if the number of command line arguments is incorrect.

Input Format
The input file, specified as the first command line argument, contains the list of processes to schedule.

All numeric values in the input file are decimal integers. The time unit doesn't matter; you can think of it as milliseconds.

Each line in the input file represents one process. Lines are sorted in increasing order of arrival time. Each line contains the following data:

```
process_name  arrival_time  total_time  cpu_burst
```

- *process_name* is a sequence of non-whitespace characters representing the name of the process.
- *arrival_time* is the time at which the process arrives in the system.
- *total_time* is the total amount of CPU time which will be used by the process.
- *cpu_burst* is amount of time the process will run before it blocks for I/O. When a process blocks, it is unavailable to run for the time specified by the command line argument *block_duration*.

Assignment

Your program will simulate the round-robin scheduling algorithm. You will need to maintain a simulation time in your program, starting at 0. You will determine the process to run according to the scheduling algorithm. You will need to determine when another process should be allowed to use the CPU. This could happen when the current process blocks, when its quantum has been used, or when the current process terminates.

Processes that enter the system or are ready to run again after a clock interrupt or blocking should be added to the end of the ready list.

Implementation notes

Process class

- The Process class constructor must have four arguments: the name of the process, the process's arrival time, the process's total running time, and the cpu burst (amount of time before the process blocks).
- Remember that all member variables should be private.
- You may add additional member variables and member functions as needed.

RR_Scheduler class

- The RR_Scheduler class constructor must have three arguments: the list of processes, the block_duration, and the quantum.
- The class must have one public member function named **run**. This function must have no parameters and must have a void return type.
- You should use private member functions to decompose your code into manageable pieces.
- It is your choice as to which data structures you use to implement the scheduler simulation.

Your main method must:

- Check to see if the correct number of command line arguments is given.
- Read the input file and create a list of processes.
- Create an object of type RR_Scheduler.
- Call the RR_Scheduler's run method to start the simulation.

Output Format (See example below)
All output should be written to standard output (the console window).

- You must first output one line containing the block_duration and the quantum as specified on the command line. These two values must be separated by a space.
- You must then output one line for each interval during which a process is running or the system is idle. The line should contain the current simulation time, the process name or [IDLE] if no process is running, the length of the interval, and a status code indicating why the interval ended: "B" for blocked, "Q" for quantum ended, "T" if the process terminated, or "I" for an idle interval.
    - Each field on the line must be separated by a tab character ("\t").
- After all processes have terminated, write the simulation time at which the last process terminated, a tab character, and the string "[END]"
- For each process you must then print the process's name, a tab character, and the process's turnaround time (one line for each process).
- Finally, you must then print the average turnaround time on a line by itself. The average should have two digits to the right of the decimal point.

Sample input file

```
P1 0 100 25
P2 100 150 30
P3 200 25 5
```

Sample output using block_duration of 30 and quantum of 20.
Your output may vary slightly depending on the order in which you add items to the ready list. In my simulation, I add processes that finish their quantum to the ready list before adding processes that arrive or processes that have completed I/O.

```
30 20
0	P1	25	B
25	[IDLE]	30	I
55	P1	25	B
80	[IDLE]	20	I
100	P2	20	Q
120	P1	20	Q
140	P2	10	B
150	P1	5	B
155	[IDLE]	25	I
180	P2	20	Q
200	P1	20	Q
220	P3	5	B
225	P2	10	B
235	P1	5	T
```

```
240     [IDLE]  15      I
255     P3      5       B
260     [IDLE]  5       I
265     P2      30      B
295     P3      5       B
300     [IDLE]  25      I
325     P2      20      Q
345     P3      5       B
350     P2      10      B
360     [IDLE]  20      I
380     P3      5       T
385     [IDLE]  5       I
390     P2      30      T
420     [END]
P1      240
P3      185
P2      320
248.333
```

Grading Criteria (35 points possible)
Note: Your program must compile to receive credit for this assignment!

Points	Criteria
0-2 points	**Program Structure:** Does the program contain the classes described above? Are function prototypes and member variables declared in the header (.h) file? Are all member variables private? Are all member functions implemented in the .cpp file of the corresponding class?
0-2 points	**Process Class:** Does the Process class constructor have four parameters as described above. Are member functions to access and update member variable implemented correctly?
0-2 points	**RR_Scheduler Class Structure:** Does the RR_Scheduler class constructor have three parameters as described above? Does it contain one public member function named "run"? Are private member function used to divide up the code into logical segments?
0-20 point	**RR_Scheduler Correctness:** Does the RR_Scheduler simulation work correctly? Do processes arrive in the system at the correct time? Do processes block at the correct times and for the correct duration? Is a new process allowed to run when the current process's quantum (time slice) ends? Do all processes run to completion?
0-5 points	**RR_Scheduler Output:** Are all output values correct? Is the output formatted as described in the assignment?
0-2 points	**Main Function:** Is the main function implemented as described in the assignment? Does it accept the three command line arguments described in the assignment? Is

	a “usage” error message printed if the number of command lines arguments is not correct? Does it correctly read process data from the input file? Does it create an instance of the RR_Scheduler and then call the run method?
0-2 points	**Style:** Is the code easy to read? Does it follow class style guidelines (See Program Style page in Canvas.)