CS162A / CSCI51.01
Group Project 1 - Process Scheduling

For this group project, you will be creating a program that simulates the different process scheduling algorithms. Given a list of processes, and a scheduling algorithm to use, you are to simulate the scheduling algorithm, and output a text version of the resulting Gantt chart (as described in the output section), as well as performance metrics after the simulation.

For the simulation, you may assume a uniprocessor setup (i.e. single processor), and that there is zero overhead (i.e. context switching is instant). In case there are any "ties" when choosing which process to run next, make it such that it will always choose the process with the smaller index. Any other "ties" should already be resolved via arrival time, assuming you are implementing an arrival queue.

Writing your program using C/C++ is preferred but not required. You may use other programming languages as long as you provide instructions on how to compile and run your program.

Input Specification

Your program should accept input via standard input (stdin). First line of input consists of a single integer T denoting the number of test cases. The next T sets of lines describes each test case.

The first line of each test case consists of an integer X (X > 0) indicating the number of processes, followed by a string S which denotes the type of scheduling algorithm that you will simulate. Possible values for S are

- FCFS (First Come First Served)
- SJF (Shortest Job First, non-preemptive)
- SRTF (Shortest Remaining Time First, SJF preemptive),
- P (Priority),
- RR (Round Robin).

In the case where S = RR, an additional integer Q (Q > 0) is provided on the same line, which denotes the time quantum.

The next X lines describes the processes (one process per line). Each line contains three integers A (A >= 0), B (B > 0), and P (-20 <= P <= 20), which denotes the arrival time, burst time, and priority value (or niceness level) of the process respectively. Note that for each test case, assume that the start time is at 0ns, but it is possible that processes may not have arrived yet. Also note that process indices are 1-based (starts from 1, not 0), i.e., the first process has a process ID of 1.

Sample input:

Input	**Explanation**
`2`	2 test cases
`4 SRTF`	Start of first test case: 4 processes, SRTF
`0 50 2`	Descriptions of the 4 processes (one process per line). The first process has a process index of 1.

`40 2 3`	Three integers describe each process: arrival time, burst time, and priority/niceness level
`20 3 1`	
`30 55 1`	
`2 FCFS`	Start of second test case: 2 processes, FCFS
`100 10 1`	Description of the 2 processes
`10 70 1`	

Output Specification

Your program should print output via standard output (stdout). For each test case, output a text version of the resulting Gantt chart as well as the performance metrics for the simulation. The first line consists of a single integer denoting the test case number (starting with 1) followed by the scheduling algorithm being used. This is then followed by a sequence of "blocks", denoting the order the processes are run. 1 block = 1 line of output.

Each block consists of three integers: the time elapsed so far in ns (nanoseconds), the process index (1-based), and the CPU time used in ns (nanoseconds). If the process has been completed at this block, append an 'X' immediately at the end of this line.

Sample output based on the sample input above:

Output	**Explanation**
`1 SRTF`	Start of output for test case 1, which uses SRTF
`0 1 20`	At the start of the simulation, we have P1 arriving in the system at time 0 with a 50ns burst time, and it runs for 20 ns. Process running: P1 Processes waiting:
`20 3 3X`	20 ns later, P3 arrives with an initial burst time of 3ns. At this point, P1 has 30 ns of burst time remaining, which means P1 gets preempted in favor of P3, which runs for 3 ns. Since P3 finished execution, we append an 'X' at the end Process running: P3 Processes waiting: P1 (30ns)
`23 1 17`	At time 23, after P3 is done, so we go back to P1 and run it for 17 ns. Note that at time 30, P4 arrives with a 55 ns burst time. No preemption happens because P1's remaining burst time is smaller. Process running: P1 Processes waiting: P4 (55ns)

`40 2 2X`	At time 40, P2 arrives with a 2ns burst time. At this point, P1 still has 13ns of burst time left, so P1 gets preempted in favor of P2. P2 runs for 2ns and finishes, hence the 'X' at the end. Process running: P2 Processes waiting: P1 (13ns), P4 (55ns)
`42 1 13X`	At time 42, P2 finishes. Since P1 has a smaller remaining time than P4, P1 continues execution and finishes after 13ns. Process running: P1 Processes waiting: P4
`55 4 55X`	At time 55, P1 finishes, and since P4 is the only one remaining, P4 starts execution and finishes after 55 ns. All processes have finished executing, so this ends the first test case.
`2 FCFS`	Start of output for test case 2, which uses FCFS
`10 2 70X`	At time 10, P2 arrives first, so we run P2 for 70ns until completion. Note that we didn't start the output at time 0 since there were no processes arriving at that time.
`100 1 10X`	At time 100, P1 arrives. Since there were no processes running, P1 starts execution and finishes after 10ns.

Note that in the case of round-robin, the behavior must mimic FCFS except that:

- Preempted processes are moved to the tail end of the queue
- New arrivals line up in a separate queue and are always given priority over the other processes that have already been given CPU time

Sample input/output for round-robin:

Input	**Output**	**Explanation**
`1`	`1 RR`	
`4 RR 25`	`0 1 25`	At time 0, P1 arrives with a 30ns burst time, so P1 starts execution for 25ns. At time 25, P2 arrives so we put P2 in the arrival queue. Process running: P1 Arrival queue: P2 (45ns) Round-robin queue: None
`0 30 3`	`25 2 25`	At time 25, P1 is preempted because its execution time is up (time quantum = 25ns) and gets sent to the round-robin queue. P2 is the next process in line in the arrival queue, so we run P2 next for 25ns. No processes arrive while P2 is running. Process running: P2 Arrival queue: None

		Round-robin queue: P1 (5ns)
25 45 4	50 1 5X	At time 50, P2's time is up so it gets preempted and sent to the tail of the round-robin queue (after P1). There are no new arrivals, so we get the next in line in the round-robin queue (P1) and run it for 5ns, finishing P1's execution. At time 55, P4 arrives, so we put P4 in the arrival queue. Process running: P1 Arrival queue: P4 (15ns) Round-robin queue: P2 (20ns)
75 10 1	55 4 15X	At time 55, P1 finishes execution. P4 gets priority over P2 since P4 is a new arrival, so we run P4 for 15ns, finishing its execution. No new processes arrive while P4 is running. Process running: P4 Arrival queue: None Round-robin queue: P2 (20ns)
55 15 5	70 2 20X	At time 70, P4 finishes execution. There are no new arrivals, so we run the next in line in the round-robin queue, which is P2. P2 runs for 20ns and finishes execution. At time 75, P3 arrives and is placed at the arrival queue. Process running: P2 Arrival queue: P3 (10ns) Round-robin queue: None
	90 3 10X	At time 90, P2 finishes execution. P3 is a new arrival, so it gets executed next and finishes execution after 10ns. No more processes are expected to arrive, which marks the end of the simulation.

Aside from the Gantt chart, you will also be printing out performance metrics for the simulation, e.g., the total time elapsed, CPU utilization, throughput, etc. Kindly refer to the provided sample output files for the information that needs to be printed, as well as the format.

Submission

Kindly archive the following into a single .zip file:

- Program source files
- A readme file on how to compile and run your program. This includes software that we need to install to run your program
- A filled in Certificate of Authorship

Name your .zip file in the following format:

- **[IDNumber1]-[IDNumber2]-[IDNumber3]-GP1.zip**
- Examples:
  - 100000-GP1.zip (solo)

- 100000-100001-GP1.zip (pair)
- 100000-100001-100002-GP1.zip (trio)

Submit your .zip file to the provided submission link, and only one person from the group needs to do the submission. Make sure that your code works and compiles properly before submitting.

Peer Evaluations

(For groups of 2 or 3 only) After submitting the group project, you will also be asked to submit peer evaluations to rate the contributions of each group member on the project. Instructions for this will be posted at a later date.