```
/* CMPUT 201 (A2 + A3) Assignments                                    */
/* Dues:                                                              */
/* #1:    11:55pm, October 8, 2018;                                   */
/* #2:    11:55pm, November 12, 2018;                                 */
/* #3:    11:55pm, December 7, 2018                                   */
```

(Mandatory assignment cover-sheet; without it, your work will not be marked.)

Submitting student: _____

Collaborating classmates: _____

Other collaborators: _____

References (excluding textbook and lecture slides):_____

_____

_____

_____

Regardless of the collaboration method allowed, you must always properly acknowledge the sources you used and people you worked with. Your professor reserves the right to give you an exam (oral, written, or both) to determine the degree that you participated in the making of the deliverable, and how well you understand what was submitted. For example, you may be asked to explain any solution that was submitted and why you choose to write it that way. This may impact the mark that you receive for the deliverable.

So, whenever you submit a deliverable, especially if you collaborate, you should be prepared for an individual inspection/walkthrough in which you explain what every line of assignment does and why you choose to write it that way.

The following is from a file named "`instance5_10_1.txt`" (also posted in eClass under Week 1):

```
#instance5_10_1.txt
#number of machines in stage 1
5
#number of jobs
10
#job processing times
31 10 28
30 25 31
8 29 28
18 3 18
1 4 3
1 31 12
30 17 27
23 16 31
2 2 1
14 14 24
#end of instance
```

This file describes an instance of two-stage open-shop with 5 parallel identical machines in stage 1 and a two-machine flow-shop as the stage 2. There are 10 jobs in the instance, each needs to be processed either first non-preemptively by a machine in stage 1 and then non-preemptively by the two-machine flow-shop, or first by the two-machine flow-shop and then by a machine in stage 1. In the following, the 5 machines in stage 1 are denoted as $A_1, A_2, \ldots, A_5$, and B and C denote the two machines in the two-machine flow-shop.

When a job is processed by the two-machine flow-shop, it has to be processed by B first, and then by C. Note that a job is not allowed to be first processed by B, then jumps out to be processed by a machine in stage 1, and lastly goes back to the flow-shop to be processed by C. In other words, assume the job chooses A_2 from the stage 1, then the machine order can only be either $\langle A_2, B, C\rangle$ or $\langle B, C, A_2\rangle$.

In the instance file, every line starting with a symbol "`#`" is a comment line; the last 10 non-comment lines each describes the non-negative integral processing times of a job on any stage 1 machine, the machine B, and the machine C, respectively. The goal of the three assignments together is to find a schedule to process all the jobs such that the latest job completion time (called the *makespan*) is as small as possible. Assignment #1 mostly deals with input and output through `stdin` and `stdout`.

In all three assignments, the instances all follow the file format shown in the above, except that the values of the variables can be different and the comment lines can vary. An instance can be written into a file, in such a case the file name convention is to start with "`instance`", followed by the number of machines in stage 1, an underscore, then the number of jobs, an underscore, and an index, and lastly the file suffix "`.txt`". That is, "`instanceX_Y_Z.txt`" is the `Z`-th instance having `X` machines in stage 1 and `Y` jobs; for example, the above "`instance5_10_1.txt`" is the first instance having 5 machines in stage 1 and 10 jobs.

The following list contains the specifications for Assignment #1 (10 marks in total):

1. Write a single program with multiple functionalities (i.e. objectives), using the command-line options. There are two options: `-i` and `-r`, which stand for reading an instance and generating random instances, respectively. Running your program without or with any other option is considered incorrect. Suppose your program name is "`myprogram`". If a command for running your program is incorrect (such as invalid options), your program prints out the following and then quits (functionality #1):

   ```
   >myprogram -i | -r
   ```

2. Option `-i` (functionality #2) is to read in an instance. To do so, you will execute the command:

```
>myprogram -i
```

Your program provides an interface to read in an instance, as follows:

```
Enter the number of machines in stage 1:
Enter the number of jobs:
Enter in each line the processing times for a job:
```

Assume the user inputs are

```
Enter the number of machines in stage 1: 5
Enter the number of jobs: 10
Enter in each line the processing times for a job: 31 10 28
30 25 31
8 29 28
18 3 18
1 4 3
1 31 12
30 17 27
23 16 31
2 2 1
14 14 24
```

Your program might encounter an error during reading, and should report "`Error in reading the instance!`" and quit; otherwise, it continues to the next item.

3. After successfully reading in an instance, your program will print (functionality #3) out the instance to the screen, with the comment lines added as in the following; and your program terminates.

```
#number of machines in stage 1
5
#number of jobs
10
#job processing times
31 10 28
30 25 31
8 29 28
18 3 18
1 4 3
1 31 12
30 17 27
23 16 31
2 2 1
14 14 24
#end of instance
```

4. Option `-r` (functionality #4) is to generate random instances. To do so, you will execute the command:

```
>myprogram -r
```

Your program provides an interface as follows:

```
Generating random instances ...
Enter the number of machines in stage 1:
Enter the number of jobs:
Enter the processing time interval [t_1, t_2]:
Enter the number of instances to be generated:
```

Assume the user inputs are

```
Generating random instances ...
Enter the number of machines in stage 1: 5
Enter the number of jobs: 10
Enter the processing time interval [t_1, t_2]: 1 32
Enter the number of instances to be generated: 4
```

Your program will generate in total 4 instances, in which each randomly generated processing time is an integer in the interval [1, 32], and print them out one after another separated by two blank lines, as in the following; and your program terminates.

```
#instance5_10_1.txt
#number of machines in stage 1
5
#number of jobs
10
#job processing times
31 10 28
30 25 31
8 29 28
18 3 18
1 4 3
1 31 12
30 17 27
23 16 31
2 2 1
14 14 24
#end of instance


#instance5_10_2.txt
#number of machines in stage 1
5
... (your printing continues on)
```

//End of description of Assignment #1.