Printer

After making a photo album editor for Rar the Cat in sit-in lab #03, Rar the Cat now thinks very highly of you and wants to give you a new task – coding the software that manages the printing of albums. The printing of a photo album involves sending a print job to the printer. As the printer can only print 1 page per 1 unit time, there is a need for a printing *queue* when there is a large number of print jobs in a short time duration.

Print jobs can be sent by any **user** (identified by **username**) and will contain a list of **K** photos to be printed. It will also be denoted by a unique timestamp **T**, which is the time the print job was sent. This means that the print job is sent exactly at the start of **T** time units since the printer was switched on (at 0 time unit). You are guaranteed that no two print jobs will have the same value of **T**. Print jobs within the **same** print queue should be printed in a First-In-First-Out manner (i.e. in increasing value of **T**).

Some printing services print a cover page for every print job. Similarly, you are to generate the cover page for each print job too. The cover page should contains details such as **username** of the user that sent the print job, **printing time** and **number of pages** of the print job. Also, they should be printed *first*, before the pages in the print job. The **printing time** of a print job is the time the cover page *started* to print. (See example for more details.)

At the end of the day, Rar the Cat wants your program to output the *stack* of pages that the printer would have printed, from top to bottom. Each page will either be a *cover page* or a *photo* and they are all printed on a single side only. The printer adds newly printed pages to the top of the stack of existing printed pages. Hence, a page that is printed first should appear lower in the stack as compared to pages that are printed later.

However, Rar the Cat wants to give priority to *premium users* and let them have priority in printing. To do so, Rar the Cat wants you to code **2 print queues**: *standard* and *priority*. Print jobs in the *priority* print queue will have priority over those in the *standard* print queue. As such, whenever the printer is able to process a new print job, it **must** process those in the *priority* print queue first. Only when the *priority* print queue is empty, then the printer can process print jobs in the *standard* print queue. Within the same print queue, print jobs are still ordered in a First-In-First-Out manner (i.e. increasing value of **T**). Do note that you can assume that the printer will be able to check and process the print job instantaneously. (i.e. a print job that is queued at time **T** can be printed instantly at time **T** if there are no other jobs in the queue and the printer is not printing anything.)

However, print jobs will not be interrupted halfway in favour of a higher priority print job. This means, if the printer is **in the middle** of processing a print job from the *standard* print queue and a new print job joins the *priority* print queue, it **will not** stop the current print job in order to process the incoming job from the *priority* print queue. Instead, the printer will only start processing the print job from the *priority* print queue when the print job it is currently processing ends.

Luckily for you, the only *premium user* is a user with username "`proftan`". Hence, when he sends a print job to the printer, his print jobs should be inside the *priority* print queue. For all other users, their print jobs should be in the *standard* print queue.

As values of **T** can be *very large*, Ivan would like to remind you to analyse the complexity of your programs. Let the number of pages printed be **P**, if your code is of the following complexity for all inputs:

I. **O(P)** – *maximum* correctness marks is 50.
II. **O(T)** – *maximum* correctness marks is 43 out of 50.
III. **O(P^2)** or slower – *maximum* correctness marks is 40 out of 50.

Input

The first line of input will contain an integer **N**, the number of print jobs to the printer.
Then **N** blocks describing print jobs will follow. Each block will be preceded with a single blank line. We guarantee print jobs will be provided in increasing values of **T** and no two print jobs will have the same value of **T**.
The first line of every block will contain an integer **T**, followed by a string **username** and lastly another integer **K**. They denote the time that the print job was sent, the user that sent the print job and the number of photos to print for this print job respectively.
The second line in the block will contain **K** integers. Each of these integers refer to a **photoID** and they are to be printed from left to right. For instance, the left-most integer in this line will be printed before the right-most integer.

Limits

- 0 < **N** ≤ 50,000 and 0 < **K** ≤ 100,000
- In addition, we guarantee that the total number of pages to be printed by the printer will not exceed 100,000. This implies that 0 < **N** + *sum* of all **K** ≤ 100,000.
- 0 ≤ **T** ≤ 1,000,000,000 and 0 ≤ **photoID** ≤ 1,000,000,000.
- **username** will not exceed 20 characters and only contain lowercase alphabets (i.e. ‘a’ to ‘z’).

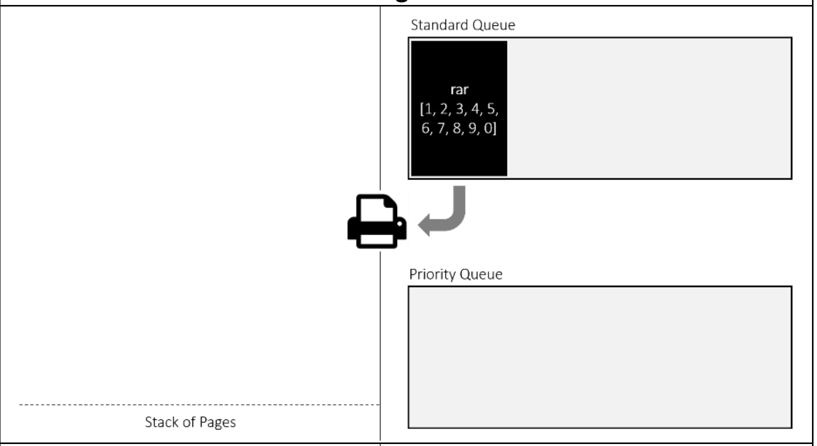
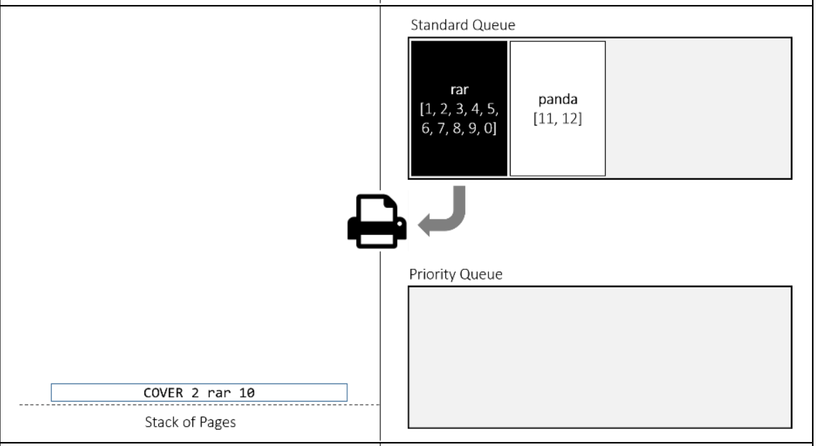
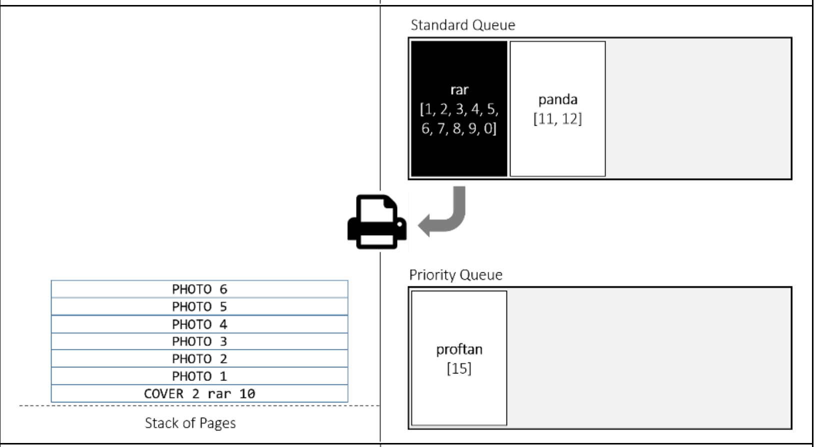
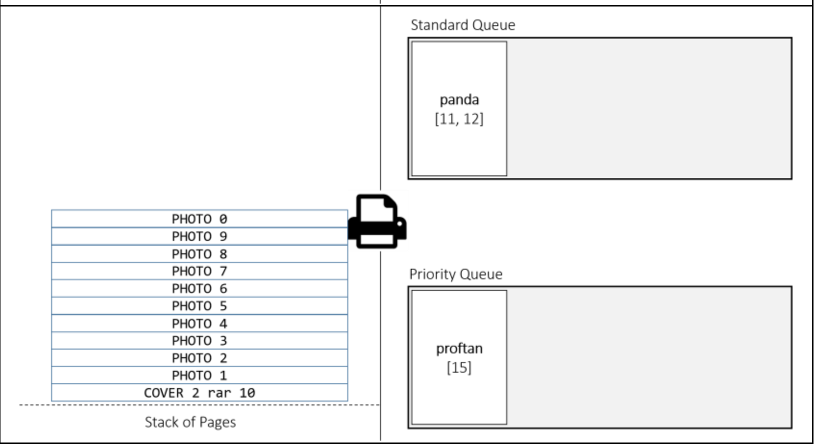
Output

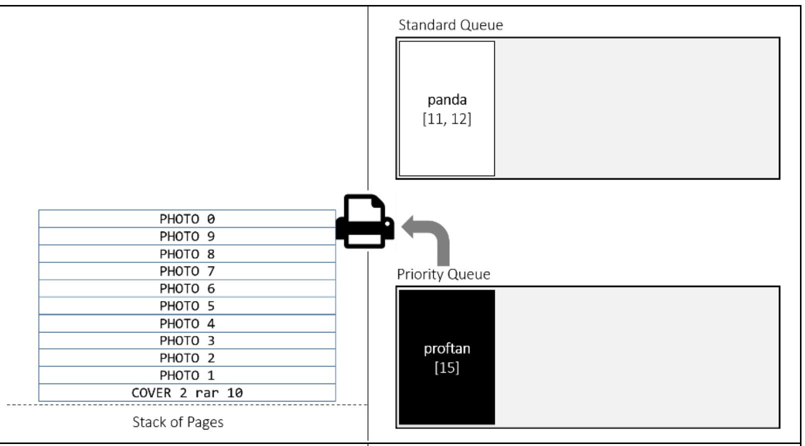
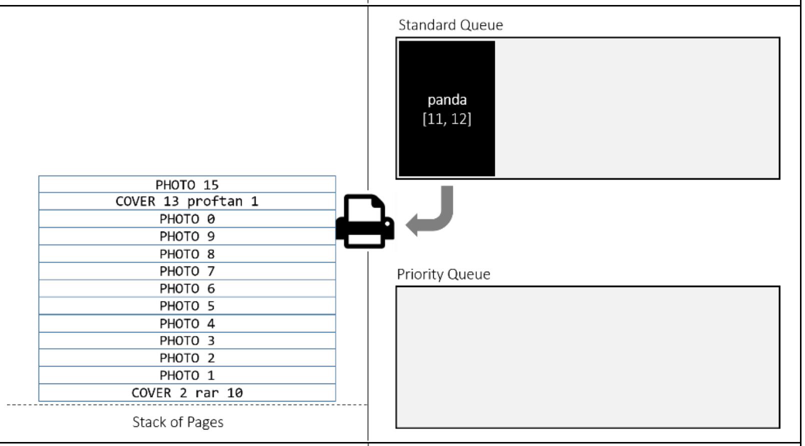
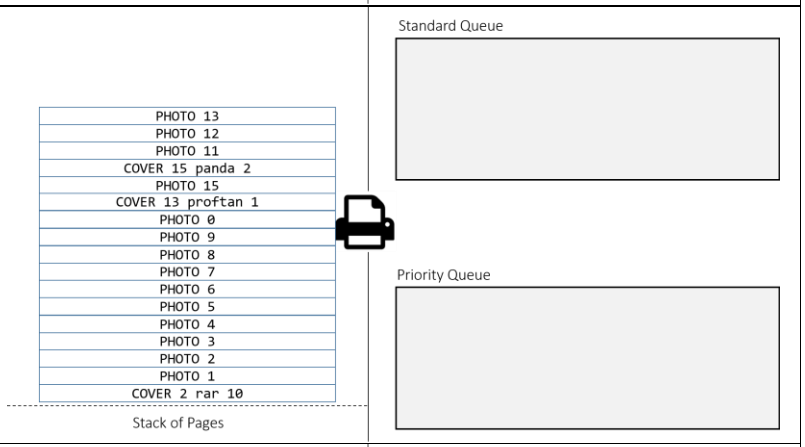
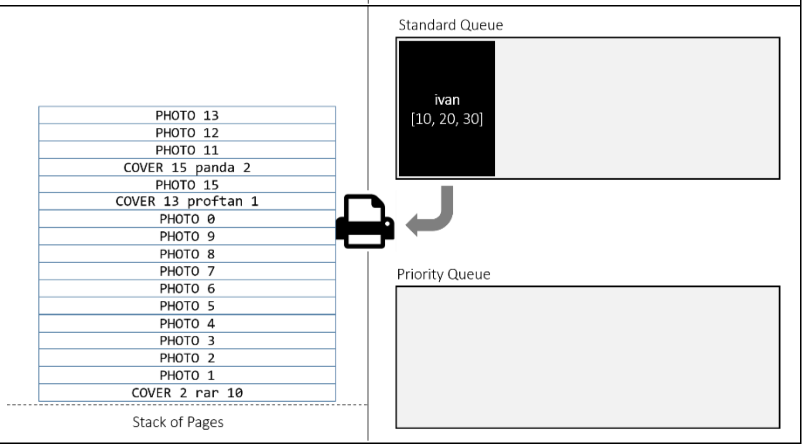
Output the *stack* of pages that the printer has printed from top to bottom.
If the page is a photo, then output “PHOTO **[photoID]**” where **[photoID]** is the ID of the photo printed.
If the page is a cover page, then output “COVER **[printing_time] [username] [number of pages printed]**” where **[printing_time]** is the time that the cover page is printed, **[username]** is the username of the user that sent the print job and **[number of pages printed]** is the value of **K** in the print job. (Do note that the number of pages printed *exclude* the cover page.)

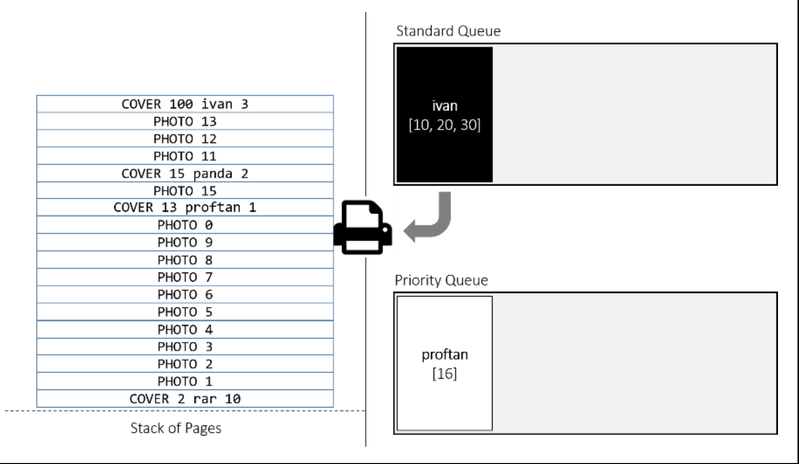
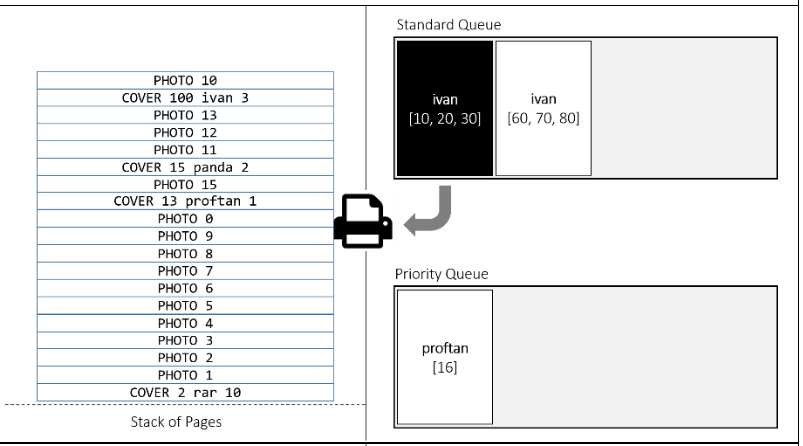
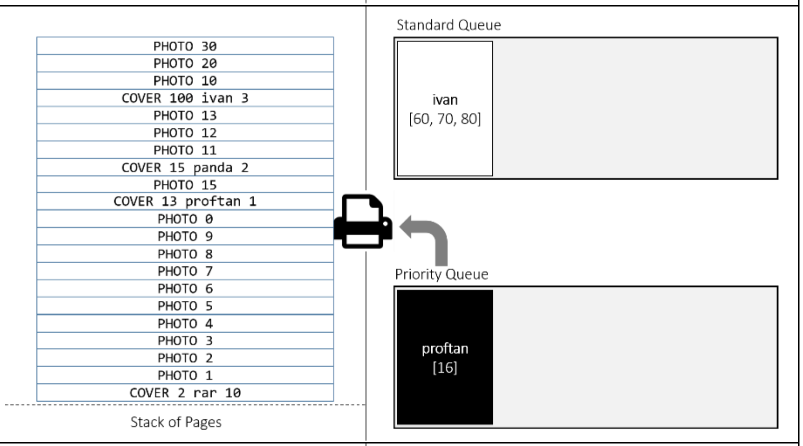
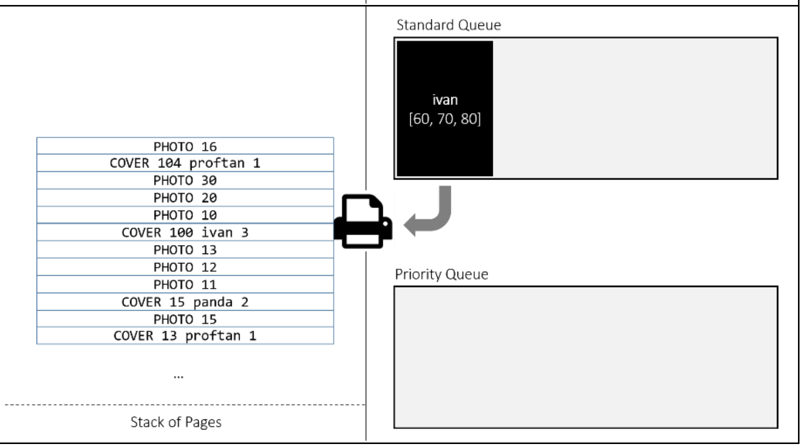
Sample Testcase

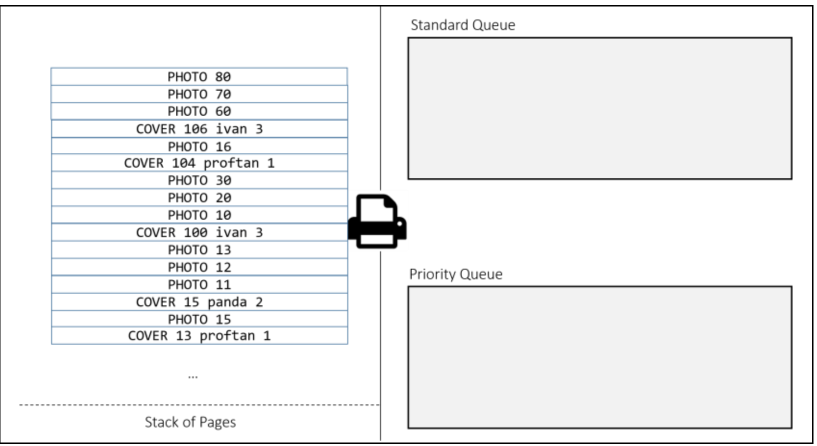
Sample Input (**printer1.in**)	Sample Output (**printer1.out**)
`6`	`PHOTO 80`
	`PHOTO 70`
`2 rar 10`	`PHOTO 60`
`1 2 3 4 5 6 7 8 9 0`	`COVER 106 ivan 3`
	`PHOTO 16`
`3 panda 2`	`COVER 104 proftan 1`
`11 12`	`PHOTO 30`
	`PHOTO 20`
`9 proftan 1`	`PHOTO 10`
`15`	`COVER 100 ivan 3`
	`PHOTO 12`
`100 ivan 3`	`PHOTO 11`
`10 20 30`	`COVER 15 panda 2`
	`PHOTO 15`
`101 proftan 1`	`COVER 13 proftan 1`
`16`	`PHOTO 0`
	`PHOTO 9`
`102 ivan 3`	`PHOTO 8`
`60 70 80`	`PHOTO 7`
	`PHOTO 6`
	`PHOTO 5`
	`PHOTO 4`
	`PHOTO 3`
	`PHOTO 2`
	`PHOTO 1`
	`COVER 2 rar 10`

Explanation for Sample Testcase

Time	Description	Diagram
2	At time 2, user "rar" submits a new print job of 10 photos. As he is not a premium user, the print job is placed in the *standard queue*. Since the printer is idle, it immediately starts on the new print job and will continue printing it for the next 11 seconds.	 Standard Queue rar [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] Priority Queue Stack of Pages
3	At time 3, user "panda" submits a new print job with 3 photos. He is not a premium user as well, hence the print job joins the *standard queue* as well. Meanwhile, the printer has finished printing the cover page of the first print job.	 Standard Queue rar [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] panda [11, 12] Priority Queue COVER 2 rar 10 Stack of Pages
9	At time 9, user "proftan" submits a new print job with 1 photo. Since "proftan" is a premium user, this print job joins the priority queue. However, since the printer is already processing a print job, it does not immediately gets printed.	 Standard Queue rar [1, 2, 3, 4, 5, 6, 7, 8, 9, 0] panda [11, 12] Priority Queue proftan [15] PHOTO 6 PHOTO 5 PHOTO 4 PHOTO 3 PHOTO 2 PHOTO 1 COVER 2 rar 10 Stack of Pages
13	At the start of time 13, the printer finishes printing the first print job. Notice that pages that are printed first, appear lower in the stack of pages. This is because the printer adds newly printed pages to the top of the stack.	 Standard Queue panda [11, 12] Priority Queue proftan [15] PHOTO 0 PHOTO 9 PHOTO 8 PHOTO 7 PHOTO 6 PHOTO 5 PHOTO 4 PHOTO 3 PHOTO 2 PHOTO 1 COVER 2 rar 10 Stack of Pages

13	As there is a print job in the priority queue, the printer will decide to process that first, before those in the standard queue. In this case, it will process the print job containing photo 15.	 Standard Queue panda [11, 12] Priority Queue proftan [15] PHOTO 0 PHOTO 9 PHOTO 8 PHOTO 7 PHOTO 6 PHOTO 5 PHOTO 4 PHOTO 3 PHOTO 2 PHOTO 1 COVER 2 rar 10 Stack of Pages
15	The printer finishes the print job right before time 15. It now processes the print job from the standard queue, since there is no more print jobs left in the priority queue.	 Standard Queue panda [11, 12] Priority Queue PHOTO 15 COVER 13 proftan 1 PHOTO 0 PHOTO 9 PHOTO 8 PHOTO 7 PHOTO 6 PHOTO 5 PHOTO 4 PHOTO 3 PHOTO 2 PHOTO 1 COVER 2 rar 10 Stack of Pages
19	The printer finishes the print job right before time 19. It now remains idle since there are no print jobs in both queues.	 Standard Queue Priority Queue PHOTO 13 PHOTO 12 PHOTO 11 COVER 15 panda 2 PHOTO 15 COVER 13 proftan 1 PHOTO 0 PHOTO 9 PHOTO 8 PHOTO 7 PHOTO 6 PHOTO 5 PHOTO 4 PHOTO 3 PHOTO 2 PHOTO 1 COVER 2 rar 10 Stack of Pages
100	At time 100, user "ivan" submits a print job with 3 photos. As he is not a premium user, the print job is added to the *standard queue*. The printer immediately begins printing it.	 Standard Queue ivan [10, 20, 30] Priority Queue PHOTO 13 PHOTO 12 PHOTO 11 COVER 15 panda 2 PHOTO 15 COVER 13 proftan 1 PHOTO 0 PHOTO 9 PHOTO 8 PHOTO 7 PHOTO 6 PHOTO 5 PHOTO 4 PHOTO 3 PHOTO 2 PHOTO 1 COVER 2 rar 10 Stack of Pages

101	At time 101, user "proftan" submits another print job with 1 photo. As "proftan" is a premium user, his print job is added to the *premium queue*. However, since the printer is currently processing another print job, the print job in the *premium queue* does not immediately gets printed.	 Standard Queue ivan [10, 20, 30] Priority Queue proftan [16] COVER 100 ivan 3 PHOTO 13 PHOTO 12 PHOTO 11 COVER 15 panda 2 PHOTO 15 COVER 13 proftan 1 PHOTO 0 PHOTO 9 PHOTO 8 PHOTO 7 PHOTO 6 PHOTO 5 PHOTO 4 PHOTO 3 PHOTO 2 PHOTO 1 COVER 2 rar 10 Stack of Pages
102	At time 102, user "ivan" submits *another* print job with 3 photos. As he is not a premium user, the print job is added to the *standard queue*.	 Standard Queue ivan [10, 20, 30] ivan [60, 70, 80] Priority Queue proftan [16] PHOTO 10 COVER 100 ivan 3 PHOTO 13 PHOTO 12 PHOTO 11 COVER 15 panda 2 PHOTO 15 COVER 13 proftan 1 PHOTO 0 PHOTO 9 PHOTO 8 PHOTO 7 PHOTO 6 PHOTO 5 PHOTO 4 PHOTO 3 PHOTO 2 PHOTO 1 COVER 2 rar 10 Stack of Pages
104	Right before time 104, the printer finishes printing ivan's print job. Now, the printer will process proftan's print job in the *priority queue*.	 Standard Queue ivan [60, 70, 80] Priority Queue proftan [16] PHOTO 30 PHOTO 20 PHOTO 10 COVER 100 ivan 3 PHOTO 13 PHOTO 12 PHOTO 11 COVER 15 panda 2 PHOTO 15 COVER 13 proftan 1 PHOTO 0 PHOTO 9 PHOTO 8 PHOTO 7 PHOTO 6 PHOTO 5 PHOTO 4 PHOTO 3 PHOTO 2 PHOTO 1 COVER 2 rar 10 Stack of Pages
106	Right before time 106, the printer finishes printing proftan's print job. Now, the printer will process jobs from the *standard queue* as the *priority queue* is empty.	 Standard Queue ivan [60, 70, 80] Priority Queue PHOTO 16 COVER 104 proftan 1 PHOTO 30 PHOTO 20 PHOTO 10 COVER 100 ivan 3 PHOTO 13 PHOTO 12 PHOTO 11 COVER 15 panda 2 PHOTO 15 COVER 13 proftan 1 ... Stack of Pages

110	Right before time 110, the printer finishes the last print job in this testcase.	 Standard Queue PHOTO 80 PHOTO 70 PHOTO 60 COVER 106 ivan 3 PHOTO 16 COVER 104 proftan 1 PHOTO 30 PHOTO 20 PHOTO 10 COVER 100 ivan 3 PHOTO 13 PHOTO 12 PHOTO 11 COVER 15 panda 2 PHOTO 15 COVER 13 proftan 1 ... Priority Queue Stack of Pages

Notes:

1. You should develop your program in the subdirectory `ex1` and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. If your algorithm is different from the given skeleton, you are free to write a solution according to your own algorithm. However, **your algorithm must use stack and/or queue** to solve this problem. You are **not allowed** to use arrays, ArrayList, HashMap, etc. for this problem.
3. For Java API Stack, you are restricted to use onlythe following functions: **push, add, pop, peek, empty, isEmpty, size, clone, clear.**
4. For Java API Queue, you are to use an object of type *LinkedList<…>* and assign it to a variable of type *Queue*<…>. [Eg: *Queue<Integer> q = new LinkedList<Integer>();*] Also, you are restricted to use only the following functions: **add, offer, peek, poll, remove, isEmpty, size, clear.**
5. You can write your own implementation of stack and/or queue. However, you are **also restricted** to using functions with *similar functionality* to the functions above, which are the standard functions of a stack and/or queue.
6. You are **free to define your own classes (or remove existing ones)** if it is suitable.
7. You **are allowed** concatenate and use strings but **only for input and output purposes**. (Eg: You can concatenate and store strings that is to be printed later in your program.)
8. Please be reminded that the marking scheme is:

Input & Output	: 20% (10% each)
Correctness	: 50%
Programming Style	: 30% (awarded if you score **at least 20% from the above**):

   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity (modular programming, proper modifiers [public / private]): 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

Compilation Error	**:** Deduction of **50% of the total marks obtained**.
Violation of Restrictions	: Deduction of up to **100% of the total marks obtained**.

Skeleton File – Printer.java

You are given the skeleton file `Printer.java`. You should see a non-empty file when you open the skeleton file. Otherwise, you might be in the wrong working directory.

You should see the following contents when you open the skeleton file:

```java
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
import java.util.Stack;
public class Printer {
    private void run() {
        //implement your "main" method here
    }
    public static void main(String[] args) {
        Printer newPrinter = new Printer();
        newPrinter.run();
    }
}
```