Birla Institute of Technology and Science, Pilani Hyderabad Campus

CS F211 Data Structures and Algorithms Practice Sheet 1: Arrays, Recursions, Linked Lists

Allowed languages: **C++**

General Tips

- Try to use functions as much as possible in your code. Functions increase reusability and the pass-by-value feature provides a significant help sometimes. Modularizing your code also helps you to debug efficiently.
- Use only the STL libraries. No other external libraries will be allowed. If you are working with the GCC compiler, a common line to add at the top of your code file is:

  ```
  #include <bits/stdc++.h>
  ```

  This will include all the required files that you will need for the purpose of the labs. It is also recommended to find out which files are included in this for your own understanding.
- Indent your code appropriately and use proper variable names, if not already provided in the question. Also, use comments wherever necessary.
- Make sure to debug your code after every task or after every code block to avoid having to debug your entire file at once in the last minute.
- Use a proper IDEs or text editors like Sublime Text or VSCode (Or Vim if you're feeling adventurous) as they help to run and test your code on multiple test-cases easily. You can install Windows Subsystem Linux (WSL) or MinGW, if you are Windows user to compile and run your programs. Alternatively, you can run and test your codes on Online GDB.

Changelog

- The statement of problem E was rephrased to make it easier to understand and implement.

A: Pest Control

There has been a mass breakout of COVID and the administration has left you in charge of quarantine room allotment for a particular wing consisting of k rooms arranged in a linear fashion. You have to try and fit k snakes and k regular people into these rooms. Since there are a total of $2 \times k$ people, exactly k people will be given a room, and k people will be left over. However, you know that if you allot two snakes in adjacent rooms, they will end up plotting some shady stuff together, and you would like to avoid that at any cost. Print all possible room allotments such that no two snakes are ever in adjacent rooms. Print them as strings of length k consisting of only the letters 'S' for snake and 'R' for regular person.
Hint: Try using recursion

Input

The first and only line of input contains a single integer k, the number of rooms in the wing $(1 \leq k \leq 8)$.

Output

In the first line, print a single integer n representing the number of possible room allotments. Each of the following n lines should contain a string of length k consisting of only the letters 'S' and 'R' as described in the question. In each of these strings, no two 'S's should be adjacent. Print the possible allotments in any order.

```
input
3

output
5
RRR
RRS
RSR
SRR
SRS

explanation
Out of the 8 possible allotments, these 5 are the only ones that satisfy the required
condition that no two snakes are next to each other.
```

B: Replacing DaSH

The scientist Okabe Rintaro has moved to the sigma-timeline where his chief tech guy Daru has died, and he needs your help getting back to the alpha and beta timelines. While working with the ALU of the time travel microwave, you notice that in order to perform operations with large numbers, numbers are stored in the form of a linked list, with the head being the leftmost digit and the tail being the rightmost digit when written in decimal notation. For example, the number 98272 would be stored as $9 \rightarrow 8 \rightarrow 2 \rightarrow 7 \rightarrow 2$. You have to write a function that adds 1 to this number and outputs the answer. Additionally, to save space, you must do this **without using any extra memory**, except for a few extra variables. You are not allowed to create extra arrays.

Input

The first line of input contains a single integer n, the number of digits in the number ($1 \leq n \leq 10^4$). The second line contains n space-separated integers, the i^{th} of which represents the i^{th} digit from the left of the number written in decimal format ($0 \leq digit_i \leq 9$). **Please note that you have to store this number in a linked list as mentioned above**.

Output

print a single line of space separated integers, representing the digits of the number after adding 1 to it.

---

input

```
9 9 9 9
```

output

```
1 0 0 0 0
```

explanation

after adding 1 to 9999, we get 10000. This addition must be done using a linked list to store the digits of the number.

C: Short People Issues

You are standing in a line and want to see what's going on at the front. Your height is x and you are currently at the very end of the line. You want to rearrange the line so that you can cleanly chop off the heads of everyone taller than you. Note that you can rearrange positions of everyone except you, and after the rearrangement, you will still be at the back of the line. There are n people in the line (not including you), and you want to rearrange them such that all people with a height less than or equal to x are at the front, and all people taller than x are at the back **without losing the original relative ordering in the two parts of the final array**. This means that, if a_i and a_j are the heights of two people and $a_i, a_j \leq x$ or $a_i, a_j \geq x$, then if a_i comes before a_j in the original array, the same must be true for the rearranged array. For example, if the heights of people from front to back are $[5, 3, 9, 11, 1, 2]$ and your height is 4, then after rearranging, it will become $[3, 1, 2, 5, 9, 11]$. The array $[1, 2, 3, 5, 9, 11]$ is incorrect because it does not maintain the relative ordering. Print the rearranged array as described. **Note that you cannot use extra space**, except for a few extra variables with low memory. You cannot use extra arrays, containers, etc., except for storing the initial height array.
Hint: You might be able to reduce space consumption by using a Linked List.

Input

The first line contains two integers, the number of people in the line, not including you, and x, your height ($1 \leq n \leq 10^4$, $1 \leq x \leq 10^9$). The next line contains n integers, representing the heights of the people in line ($1 \leq a_i \leq 10^9$).

Output

Print a single line of n integers, representing the rearranged array. You must do this without using extra space.

```
input
7 9
8 17 9 9 12 2 3

output
8 9 9 2 3 17 12

explanation
```

The heights less than or equal to x are 8, 9, 9, 2, and 3. The heights greater than x are 17 and 12, all in that order. Since it is important to maintain order, this is the correct array.

D: Stars and Bars

You are given a number consisting of digits between 1 and 9 inclusive. You can split the number into multiple numbers that must each contain at least one digit. For example, you can split 45678 into 4, 567, and 8. You can also split it into 456 and 78. It is also allowed to leave the number as it is without splitting it. Print all possible splits for a given number. Note that the numbers in the same split need not be distinct.
Hint: While this may seem like a combinatorics problem, try thinking of a recursive solution. This will make printing the numbers easier.

Input

The first and only line contains a single integers given as a string with length n ($1 \leq n \leq 8$). It is guaranteed that each digit is in between 1 and 9 inclusive.

Output

In the first line, print the number of possible splits. The following lines should contain all the splits. Two numbers in a single split must be space-separated. Refer to the sample case below for more clarity. Print the splits in any order.

```
input
1895

output
8
1 8 9 5
1 8 95
1 89 5
1 895
18 9 5
18 95
189 5
1895

explanation
It can be worked out and shown that these are all the possible splits.  They can be
printed in any order.
```

E: Round Robin

The Round Robin scheduling algorithm is a widely used CPU scheduling algorithm in traditional operating systems which is used to decide which resources to allocate to which program. In this algorithm, each process is assigned a fixed time slot in a cyclic way. You are given n processes as well as the execution time and process ID of each process. In the round robin algorithm, the processes are in a queue where the first process in the queue is being executed currently. There are two cases where the execution of the current program stops:

- If the program completes its execution
- If the program attempts to run for longer than x time units

If the program successfully completes its execution, it is removed from the queue. In case the program attempts to run longer than x time units, **the program is pre-empted or paused, and is appended to the back of the queue** so that it can complete its execution at a later stage. Note that if a_i is the time needed to finish the execution of the i^{th} program and it gets pre-empted after x seconds, then the next time it runs, it only needs $a_i - x$ seconds to be completed.

For example, if the processes have an execution time of $[5, 3, 6]$ and $x = 3$, then the queue in the round robin algorithm will look something like this: $[5, 3, 6] \rightarrow [3, 6, 2] \rightarrow [6, 2] \rightarrow [2, 3] \rightarrow [3] \rightarrow []$. All processes are now completed.

Your task is to replicate a basic version of the Round Robin algorithm. At the start of every time unit, you will be given three types of queries. A query of type 0 means you do nothing. If you are given a query of type 1, you must output the process ID of the process being executed at the start of the current time unit. If there is no process being executed, print the string "All processes finished". A query of type 2 means you have to add a process to the back of the execution queue. You will be given the execution time of this process as well as its process id.

Please refrain from using an actual queue data structure. Try and use a variation of a linked list.

Input

The first line of the input contains two integers, n and x, the number of processes currently in the queue, and the time slot respectively. Two lines then follow, the first of which contains the process IDs of the n processes. It is guaranteed that the process IDs are all integers $\leq 10^6$, and all process ID's are distinct. The Third line contains the execution time of each of the n processes, all of which are ≤ 100.

In the next line, there is a single integer q, the number of queries. q lines follow, each of them is a particular type of query:

- 0
- 1

- 2 *e id*

A query of type 0 means you do nothing. A query of type 1 means that you have to output the ID of the process executing at the current time step, or print "All processes finished" if the queue is empty. A query of type 2 means that you have to add a process with ID *id* and execution time e to the back of the queue at the beginning of the current time unit.

Output

For every query of type 1, output the process running at the start of time t, or print "All processes finished" if the queue is empty.

```
input
3 3
100 200 300
5 3 6
10
0
2 4 400
0
0
1
0
0
0
0
1

output
200
400

explanation
This is the queue at the start of each time step in terms of execution times up to
10 seconds:
t=1:  [5,3,6]
t=2:  [4,3,6,4]
t=3:  [3,3,6,4]
t=4:  [3,6,4,2] (process is pre-empted and pushed to the back of the queue)
t=5:  [2,6,4,2]
t=6:  [1,6,4,2]
t=7:  [6,4,2] (the previous process is finished and removed from the queue)
t=8:  [5,4,2]
t=9:  [4,4,2]
t=10:  [4,2,3] (process is pre-empted and pushed to the back of the queue)
```