**Exercise: Stacks and Queues**

Problems for exercises and homework for the ["CSharp Advanced" course @ Software University](https://softuni.bg/courses/csharp-advanced).

You can check your solutions here: <https://judge.softuni.bg/Contests/1447/Stacks-and-Queues-Exercise>

* **Basic Stack Operations**

Play around with a stack. You will be given an integer **N** representing the number of elements to push into the stack, an integer **S** representing the number of elements to pop from the stack and finally an integer **X**, an element that you should look for in the stack. If it’s found, print "**true**"on the console. If it isn’t, print the **smallest** element currently present in the stack. If there are **no** **elements** in the sequence, print **0** on the console.

**Input**

* On the first line you will be given **N**, **S** and **X,** separated by a single space
* On the next line you will be given **N** number of integers

**Output**

* On a single line print either **true** if **X** is present in the stack, otherwise print the **smallest** element in the stack. If the stack is **empty**, print 0

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5 2 13  1 13 45 32 4 | true | We have to **push 5** elements. Then we **pop 2** of them. Finally, we have to check whether 13 is present in the stack. Since it is we print **true**. |
| 4 1 666  420 69 13 666 | 13 |  |

* **Basic Queue Operations**

Play around with a queue. You will be given an integer **N** representing the number of elements to enqueue (**add**), an integer **S** representing the **number of elements** to **dequeue** (**remove**) from the queue and finally an integer **X**, an element that you should look for in the **queue**. If it is, print **true** on the console. If it’s not print the **smallest** **element** currently present in the queue. If there are **no** **elements** in the sequence, print **0** on the console.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5 2 32  1 13 45 32 4 | true | We have to **enqueue 5** elements. Then we **dequeue** **2** of them. Finally, we have to check whether 32 is present in the queue. Since it is we print **true**. |
| 4 1 666  666 69 13 420 | 13 |  |

* **Maximum and Minimum Element**

You have an empty sequence, and you will be given **N** queries. Each query is one of these three types:

1 x – **Push** the element x into the stack.

2 – **Delete** the element present at the **top** of the **stack**.

3 – **Print** the **maximum** element in the stack.

4 – **Print** the **minimum** element in the stack.

After you go through all of the queries, print the stack in the following format:

**"{n}, {n1}, {n2} …, {nn}"**

**Input**

* The first line of input contains an integer, **N**
* The next **N** lines each contain an above-mentioned query. *(It is guaranteed that each query is valid.)*

**Output**

* For each type 3 or 4 query, print the **maximum**/minimum element in the stack on a new line

**Constraints**

* 1 ≤ N ≤ 105
* 1 ≤ x ≤ 109
* 1 ≤ type ≤ 4
* If there are **no elements** in the stack, **don’t print anything** on commands 3 and 4

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 9  1 97  2  1 20  2  1 26  1 20  3  1 91  4 | 26  20  91, 20, 26 |
| 10  2  1 47  1 66  1 32  4  3  1 25  1 16  1 8  4 | 32  66  8  8, 16, 25, 32, 66, 47 |

* **Fast Food**

You have a fast food restaurant and most of the food that you're offering is previously prepared. You need to know if you will have enough food to serve lunch to all of your customers. Write a program that checks the orders’ quantity. You also want to know the client with the **biggest** order for the day, because you want to give him a discount the next time he comes.

First, you will be given the **quantity** **of the food** that you have for the day (an integer number). Next, you will be given **a sequence of integers**, each representing the **quantity of an order**. Keep the orders in a **queue**. Find the **biggest** **order** and **print** it. You will begin servicing your clients from the **first** **one** that came. Before each order, **check** if you have enough food left to complete it. If you have, **remove the order** from the queue and **reduce** the amount of food you have. If you succeeded in servicing all of your clients, print:

**"Orders complete".**

If not, print:

**"Orders left: {order1} {order2} .... {orderN}".**

**Input**

* On the first line you will be given the quantity of your food - **an integer** in the range [0, 1000]
* On the second line you will receive a sequence of integers, representing each order, **separated by a single space**

**Output**

* Print the quantity of biggest order
* Print "Orders complete" if the orders are complete
* If there are orders left, print them in the format given above

**Constraints**

* The input will always be valid

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 348  20 54 30 16 7 9 | 54  Orders complete |
| 499  57 45 62 70 33 90 88 76 | 90  Orders left: 76 |

* **Songs Queue**

Write a program that keeps track of a songs queue. The **first** song that is put in the queue, should be the **first** that **gets played**. A song cannot be added if it is currently in the queue.

You will be given **a sequence of songs**, separated by a comma and a single space. After that you will be given **commands** **until** there are **no songs enqueued**. When there are **no more songs** in the queue **print** "**No more songs!**" and **stop** the **program**.

The possible commands are:

* "**Play**" - plays a song (removes it from the queue)
* "**Add {song}**" - adds the song to the queue if it isn’t contained already, otherwise print "**{song} is already contained!**"
* "**Show**" - prints all songs in the queue separated by a comma and a white space (start from the first song in the queue to the last)

**Input**

* On the first line, you will be given a sequence of strings, separated by a comma and a white space
* On the next lines you will be given commands until there are no songs in the queue

**Output**

* While receiving the commands, print the proper messages described above
* After the command "Show", print the songs from the **first** to the **last**

**Constraints**

* The input **will always be valid** and in the **formats** described above
* There **might** be commands **even after** there are **no songs in the queue** (ignore them)
* There will never be duplicate songs in the initial queue

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| All Over Again, Watch Me  Play  Add Watch Me  Add Love Me Harder  Add Promises  Show  Play  Play  Play  Play | Watch Me is already contained!  Watch Me, Love Me Harder, Promises  No more songs! |
| Wake Up, Senorita, Best Song Ever, I Know You  Add Best Song Ever  Add Up Wake  Show  Play  Play  Play  Play  Show  Play  Add Watch Me Whip  Play | Best Song Ever is already contained!  Wake Up, Senorita, Best Song Ever, I Know You, Up Wake  Up Wake  No more songs! |