# Mini Exam: Stacks and Queues

You can check your solutions here: <https://judge.softuni.bg/Contests/3174/Additional-Exercises>.

## Operations with Stack

You will be given an integer **N** and integer **S**. The **N** integer is the number of elements that have to be pushed into the stack. The **S** integer is the number of elements that have to be popped from the stack. Finally you will be given an integer **X**, an element that you should look for in the stack. If it’s found, print "**found**"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 "**nothing** **found**" on the console.

### Input

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

### Output

* If **X** is present in the stack, print “**found”**, otherwise print the **smallest** element in the stack. If the stack is **empty**, print "**nothing** **found**"

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 10 5 1  0 0 0 0 1 0 0 0 0 1 | found | We will **push 10** elements. After that we will **pop 5** of them. At the end, we have to check if 1 is present in the stack. If it is so we print **found**. |
| 4 1 666  420 69 17 666 | 17 |  |

## Restaurant

You have a restaurant and you need to know if you will have enough food to service all received orders. For this purpose you need a program that checks the orders’ quantity. The restaurant gives a discount for the next order to the customer with the **biggest** **order for the day**. You need to **find that order and print it**.

At first you will receive an integer number that represent the **quantity of the food** that you have for the day. After that, you will be given a **sequence of integers** in which each number is the **quantity of an order**. You better keep the orders in **queue**. You will start servicing your orders from the **first one** that came. Before you finish each order, **check** if you have enough left food for it. If you have enough food for the order, **remove** **that order** from the queue and **reduce** the amount of food you have. If you successfully serve all customer orders, print:

"All orders are completed".

If there are orders left:

"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

* On the first line, print the **biggest order**
* On the second line, if the orders are complete, print "**All orders are completed**"
* If there are **orders left, print** them on the tirth line in the format given above

### Constraints

* The input will always be valid

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 130  42 30 28 30 | 42  All orders are completed |
| 220  60 25 45 43 78 32 64 | 78  Orders left: 78 32 64 |

**\* Traffic Jam**

Tom finally found some time to go on a **holiday**. Even on his holiday trip, he is still running into **problems**. He got stuck in a traffic jam at a very active crossroads where a lot of **accidents** happen.

He asked you to report if a **crash happened** or everyone **passed** the **crossroads safely**. The road is a **line** where the cars are in queue. When the **traffic light goes green**, the cars start passing one by one, char by char until is green and during the free window. In **one second** only **one part** of a car, (a **single character** of her name) passes the crossroad. After the **free window**, if there is character from car name that did not pass the crossroad, then that car will get **hit at that character**.

### Input

* At first, you will receive the seconds of the **green** **light** – an **integer** **in the range [1-100]**
* After that, you will receive the seconds of the **free** **window** – an **integer** **in the range [0-100]**
* Until you receive the command "**END**", you will receive one of two things:
  + A **car name** –**string** containing ASCII characters, or
  + The "**green**" command, which indicates the **start** of a **green** **traffic light**

A **green traffic** **light** goes as follows:

* If it is **green** **light** cars will enter and exit the crossroads one by one
* On the **free window** cars will only exit the crossroads

### Output

* If a **crash** **happens**, **end the program** and print:  
  " Crash on the crossroad!"  
  "{car} was hit at {characterHit}."
* If **no crash** happened and you receive an "**END**" command, print:  
  " No crash happened"  
  **"**{totalCarsPassed} total cars passed the crossroads.**"**

### Constraints

* The input will be **within the constaints** specified above and will **always be valid**. There is **no need** to check it explicitly.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 10  5  Maserati  green  Maserati  Kia  Mazda  green  END | No crash happened.  3 total cars passed the crossroads. | At the first green light (10 seconds), the Maserati(8 chars) passes safely.  During the second green light, the Maserati(8 chars) passes safely and there are 2 seconds left.  The Kia enters the crossroads and when the green light ends, it still has 1 part (char) inside ('a'), but has 5 seconds free window to leave successfully.  No more green light so the Mazda never enters the crossroads, so 3 cars passed successfully. |
| 9  3  Infiniti  Toyota  green  Toyota  Infiniti  green  END | Crash on the crossroad!  Toyota was hit at t. | Infiniti(8 chars) passes successfully and Toyota(6 chars) enters the crossroads but only the 'T' passes during the green light. There are 3 seconds of free window, so "oyo" passes and the car gets hit at 't' and the program ends with a crash. |