04. Linear Containers – Homework Exercises

Write C++ code for solving the tasks on the following pages.

Code should compile under the C++03 or the C++11 standard.

Please submit a single **`.cpp`** file for each task.

`.cpp` files for the tasks should be named with the task number followed by what you feel describes the exercise in a few words.

E.g. a good name for task 2 of this homework would be:
`2.positions-of.cpp`

Don't worry about the name too much, just make sure the number and the file extension are correct.

Problem 1 – Compare Matrices

Write a program that reads two integer matrices (2D arrays) from the console and compares them element by element. For better code reusability, you could do the comparison in a function, which returns **`true`** if they are equal and **`false`** if not.

Each matrix definition on the console will contain a line with a positive integer number **R** – the number of rows in the matrix – followed by R lines containing the numbers in the matrix, separated by spaces (each line will have an equal amount of numbers.

The matrices will have at most **`10`** rows and at most **`10`** columns.

Print **`equal`** if the matrices match, and **`not equal`** if they don't match.

Examples

Input	Output
1 1 2 3 1 1 2 3	equal
2 1 2 3 2 1 3 2 1 2 3 2 1 3	equal
4 1 11 21 31 4 1 11 21 31	equal
2	not equal

1 2 3 4 5 6 2 1 3 2 4 5 6	
2 1 2 3 4 5 6 2 1 2 3 4	not equal

Problem 2 – Positions Of

Write a program that reads a matrix of integers from the console, then a number, and prints all the positions at which that number appears in the matrix.

The matrix definition on the console will contain a line with two positive integer numbers **R** and **C** – the number of rows and columns in the matrix – followed by **R** lines, each containing **C** numbers (separated by spaces), representing each row of the matrix.

The number you will need to find the positions of will be entered on a single line, after the matrix.

You should print each position on a single line – first print the row, then the column at which the number appears.

If the number does not appear in the matrix, print **`not found`**

Examples

Input	Output
2 3 1 2 3 4 5 6 3	0 2
2 3 1 2 3 4 2 2 2	0 1 1 1 1 2
2 3 1 -2 -3 4 -5 6 5	not found

Problem 3 – Merge Trains

At a certain train station, trains arrive on two tracks – Track A and Track B – and are merged onto a single track. Each railcar has a number, and in each train the railcars are ordered by number – the front railcar has the lowest number, the back railcar has the highest number.

Given the railcar numbers on Track A and Track B, given from the back car to the front car, output the order in which railcars from Track A and Track B should be merged into the single track, then output the new train configuration, starting from the last railcar, and finishing at the first railcar. Railcars are moved starting from the front to the back (i.e. you can only move the front car of a train). Each time you move a railcar from the parallel tracks A and B to the merge track, it pushes forward any railcars that are already there.

The first line of the console will contain the numbers of the railcars on Track A, from the last railcar to the first railcar.

The second line of the console will contain the numbers of the railcars on Track B, from the last railcar to the first railcar.

If a railcar from Track A should be moved to the single track, print A. Otherwise, print B.

On a separate line print the final configuration of the train (numbers separated by spaces, representing the railcar numbers from the last railcar to the first).

The "input" railcars will always be correctly ordered (i.e. will be a line of descending positive integer numbers). The input will be such that the result will never have any railcars with the same numbers.

Examples

Input	Output	Explanation
11 4 2 1 5 3	AABABA 11 5 4 3 2 1	We first move from A, railcar 1, and the result becomes: 1 We again move from A, railcar 2, which pushes railcar 1 more to the right: 2 1 Now we need B (the rightmost of A is 4, the rightmost of B is 3), which again pushes the railcars in the merge: 3 2 1 We now again need A: 4 3 2 1 Back to B, that's the last railcar there: 5 4 3 2 1 Finally we move the last from A and get the result: 11 5 4 3 2 1 - the sequence was AABABA
2 1	BA 2 1	Only two railcars, line B has the smaller railcar, so move that first, then move from A

Problem 4 – Brackets

Write a program which reads a single line from the console, containing brackets for a mathematical expression (only the brackets will appear in the input), and determines whether the brackets in the expression are correct. There are 3 types of brackets – **{}**, **[]** and **()**. **{}** can contain **{}**, **[]** and **()** brackets. Square brackets **[]** can contain **[]** and **()** brackets. Curved **()** brackets can contain only **()** brackets. Said simply, each type of brackets can contain the

same type of brackets inside, or a "lower" type of brackets (**()** is lower than **[]** which is lower than **{}**). If a bracket of one type is opened, it needs to be closed before a bracket of another type is closed.
Print **valid** if the brackets in the expression are valid and **invalid** if they are not.

Examples

Input	Output
`[()]{}{[()()]()}`	`valid`
`[(])`	`invalid`
`([])`	`invalid`
`()[[[()]]]{[()]}`	`valid`