Linear Data Structures - Exercises

This document defines the exercises for "Java Advanced" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

Matrices:

1. Fill the Matrix

Filling a matrix in the regular way (**top to bottom** and **left to right**) is boring. Write two **methods** that **fill** a **matrix** of size **N x** N in **two** different **patterns.** Both patterns are described below:

Pattern A	Pattern B
1 5 9 13 2 6 10 14 3 7 11 15 4 8 12 16	1 8 9 16 2 7 10 15 3 6 11 14 4 5 12 13

Examples

Input	Output
3, A	1 4 7 2 5 8 3 6 9
3, B	1 6 7 2 5 8 3 4 9

Hints

- Make a different method for each pattern
- Make a method for printing the matrix

2. Matrix of Palindromes

Write a program to generate the following **matrix of palindromes** of **3** letters with **r** rows and **c** columns like the one in the examples below.

- **Rows** define the first and the last letter: row 0 → 'a', row 1 → 'b', row 2 → 'c', …
- **Columns + rows** define the middle letter:
  - column 0, row 0 → 'a', column 1, row 0 → 'b', column 2, row 0 → 'c', …
  - column 0, row 1 → 'b', column 1, row 1 → 'c', column 2, row 1 → 'd', …

Input

- The numbers **r** and **c** stay at the first line at the input.
- **r** and **c** are integers in the range [1…26].
- **r + c** ≤ 27

Examples

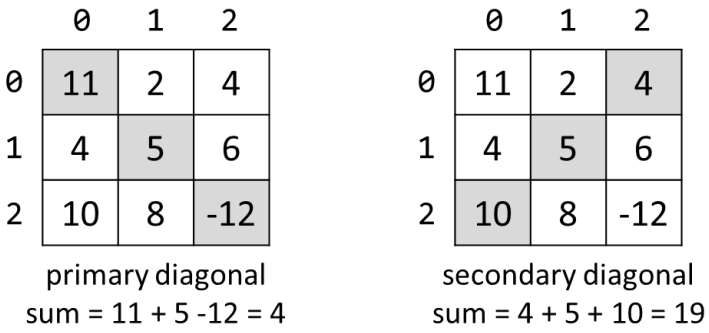
Input	Output
4 6	aaa aba aca ada aea afa bbb bcb bdb beb bfb bgb ccc cdc cec cfc cgc chc ddd ded dfd dgd dhd did
3 2	aaa aba bbb bcb ccc cdc

Hints

- Use two nested loops to generate the matrix.
- Print the matrix row by row in a loop.
- Don't forget to pack everything in methods.

3. Diagonal Difference

Write a program that finds the **difference between the sums of the square matrix diagonals** (absolute value).



Input

- The **first line** holds a number **n** – the size of matrix.
- The next **n lines** hold the **values for every row** – **n** numbers separated by a space.

Examples

Input	Output	Comments
3 11 2 4 4 5 6	15	**Primary diagonal:** sum = 11 + 5 + (-12) = 4 **Secondary diagonal:** sum = 4 + 5 + 10 = 19 **Difference:** |4 - 19| = 15

10 8 -12		

Hints

- Use a **single** loop **`i`** = **`[1 … n]`** to sum the diagonals.
- The **primary diagonal** holds all cells {**`row`**, **`col`**} where **`row`** == **`col`** == **`i`**.
- The **secondary diagonal** holds all cells {**`row`**, **`col`**} where **`row`** == **`i`** and **`col`** == **`n-1-i`**.

4. Maximal Sum

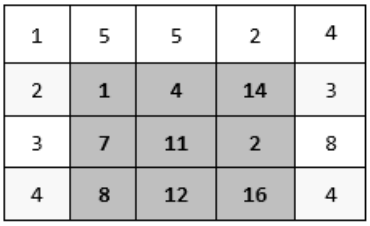
Write a program that reads a rectangular integer matrix of size **N x M** and finds in it the square **3 x 3** that **has maximal sum of its elements**.

Input

- On the first line, you will receive the rows **N** and columns **M**.
- On the next **N lines** you will receive **each row with its elements**.

Print the **elements** of the 3 x 3 square as a matrix, along with their **sum**. See the format of the output below:

Examples

Input	Output	Comments
4 5 1 5 5 2 4 2 1 4 14 3 3 7 11 2 8 4 8 12 16 4	Sum = 75 1 4 14 7 11 2 8 12 16	

5. Matrix shuffling

Write a program which reads a string matrix from the console and performs certain operations with its elements. User input is provided in a similar way like in the problems above – first you read the **dimensions** and then the **data**.

Your program should then receive commands in format: "**swap row1 col1 row2c col2**" where row1, row2, col1, col2 are **coordinates** in the matrix. In order for a command to be valid, it should start with the "**swap**" keyword along with **four valid coordinates** (no more, no less). You should **swap the values** at the given coordinates (cell [row1, col1] with cell [row2, col2]) **and print the matrix at each step** (thus you'll be able to check if the operation was performed correctly).

If the **command is not valid** (doesn't contain the keyword "swap", has fewer or more coordinates entered or the given coordinates do not exist), print "**Invalid input!**" and move on to the next command. Your program should finish when the string "**END**" is entered.

Examples

Input	Output

2 3 1 2 3 4 5 6 swap 0 0 1 1 swap 10 9 8 7 swap 0 1 1 0 END	5 2 3 4 1 6 Invalid input! 5 4 3 2 1 6
1 2 Hello World 0 0 0 1 swap 0 0 0 1 swap 0 1 0 0 END	Invalid input! World Hello Hello World

Hints

- Think about `Exception Handling`

6. * String Matrix Rotation

You are given a **sequence of text lines**. Assume these text lines form a **matrix of characters** (pad the missing positions with spaces to build a rectangular matrix). Write a program to **rotate the matrix** by 90, 180, 270, 360, … degrees. Print the result at the console as sequence of strings after receiving the "**END**" command.

Examples

Input	Rotate(90)	Rotate(180)	Rotate(270)
hello softuni exam END	e s h x o e a f l m t l u o n i	m a x e i n u t f o s o l l e h	i n o u l t m l f a e o x h s e
h e l l o s o f t u n i e x a m			

Input

The input is read from the console:

- The first line holds a command in format "**Rotate(X)**" where **X** are the degrees of the requested rotation.
- The next lines contain the **lines of the matrix** for rotation.
- The input ends with the command "**END**".

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

Print at the console the **rotated matrix** as a sequence of text lines.

Constraints

- The rotation **degrees** is positive integer in the range [0…90000], where **degrees** is **multiple of 90**.
- The number of matrix lines is in the range [1…**1 000**].
- The matrix lines are **strings** of length 1 … 1 000.
- Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

Examples

Input	Output
Rotate(90) hello softuni exam END	esh xoe afl mtl uo n i

Input	Output
Rotate(180) hello softuni exam END	maxe inutfos olleh

Input	Output
Rotate(270) hello softuni exam END	i n ou ltm lfa eox hse

Input	Output
Rotate(720) js exam END	js exam

Input	Output
Rotate(810) js exam END	ej xs a m

Input	Output
Rotate(0) js exam END	js exam

Stack and Queues:

7. Reverse Numbers with a Stack

Write a program that reads **N integers** from the console and **reverses them using a stack**. Use the **`ArrayDeque<Integer>`** class. Just put the input numbers in the stack and pop them. Examples:

Examples

Input	Output
1 2 3 4 5	5 4 3 2 1
1	1

8. Basic Stack Operations

You will be given an integer **N** representing the **number of elements to push onto 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 check whether is present in the stack**. If it is, print **true** on the console. If it's not, print the smallest element currently present in the stack.

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 a line of numbers **separated by one or more white spaces**.

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	Pop one element (666) and then check if 666 is present in the stack. It's not, so print the smallest element (13)

9. Maximum Element

You have an empty sequence, and you will be given **N** commands. Each command is one of the following 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.

Input

- The first line of input contains an integer **N**, where **$1 \leq N \leq 10^5$**
- The next **N** lines contain commands. All commands will be valid and in the format described
- The element **X** will be in range **$1 \leq X \leq 10^9$**
- The **type of the command** will be in range **$1 \leq Type \leq 3$**

Output

- For each command of **type "3"**, **print the maximum element** in the stack on a new line.

Examples

Input	Output	Comments
9 1 97	26 91	9 commands Push 97

2 1 20 2 1 26 1 20 3 1 91 3		Pop an element Push 20 Pop an element Push 26 Push 20 Print the maximum element (26) Push 91 Print the maximum element (91)

10. Basic Queue Operations

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/poll) from the queue and finally an integer **X**, an element that you should **check whether is present in the queue**. If it is print **true** on the console, if it's not **print the smallest element currently present in the queue**.

Examples

Input	Output	Comments
5 2 32 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 1 666 666 69 13 420	13	
3 3 90 90 90 90	0	

11. Robotics

Somewhere in the future, there is a robotics factory. The current project is assembly line robots.

Each robot has a **processing time**, the time it needs to process a product. When a **robot is free** it should **take a product for processing** and log his name, product and processing start time.

Each robot **processes a product coming from the assembly line**. A **product is coming** from the line **each second** (so the first product should appear at [start time + 1 second]). If a product passes the line and **there is not a free robot** to take it, it should be **queued at the end of the line again**.

The robots are **standing on the line in the order of their appearance**.

Input

- On the first line, you will get the names of the robots and their processing times in format **"robotName-processTime;robotName-processTime;robotName-processTime"**
- On the second line, you will get the starting time in format **"hh:mm:ss"**
- Next, until the **"End"** command, you will get a product on each line.

Examples

Input	Output
ROB-15;SS2-10;NX8000-3 8:00:00 detail glass wood apple End	ROB - detail [08:00:01] SS2 - glass [08:00:02] NX8000 - wood [08:00:03] NX8000 - apple [08:00:06]
ROB-60 7:59:59 detail glass wood sock End	ROB - detail [08:00:00] ROB - sock [08:01:00] ROB - wood [08:02:00] ROB - glass [08:03:00]

12. Balanced Parentheses

Given a sequence consisting of parentheses, determine **whether the expression is balanced**. A sequence of parentheses **is balanced if** every open parenthesis can be paired uniquely with a closed parenthesis that occurs after the former. Also, **the interval between them must be balanced**. You will be given three types of parentheses: (, {, and [.

{[()]} - This is a balanced parenthesis.

{[(])} - This is not a balanced parenthesis.

Input

- Each input consists of a single line, the sequence of parentheses.
- **1 ≤ Length of sequence ≤ 1000**
- Each character of the sequence will be one of the following: **{, }, (,), [,]**.

Output

- For each test case, print on a new line **"YES"** if the parentheses are balanced. Otherwise, print **"NO"**.

Examples

Input	Output
{[()]}	YES
{[(])}	NO
{{[[(())]]}}	YES

13. Recursive Fibonacci

Each member of the **Fibonacci sequence** is calculated from the **sum of the two previous members**. The first two elements are 1, 1. Therefore the sequence goes as 1, 1, 2, 3, 5, 8, 13, 21, 34…

The following sequence can be generated with an array, but that's easy, so **your task is to implement it recursively**.

If the function **getFibonacci(n)** returns the n^{th} Fibonacci number, we can express it using **getFibonacci(n) = getFibonacci(n-1) + getFibonacci(n-2)**.

However, this will never end and in a few seconds a Stack Overflow Exception is thrown. In order for the recursion to stop it has to have a "bottom". The bottom of the recursion is getFibonacci(1), and should return 1. The same goes for getFibonacci(0).

Input

- On the only line in the input the user should enter the wanted Fibonacci number N where 1 ≤ N ≤ 49

Output

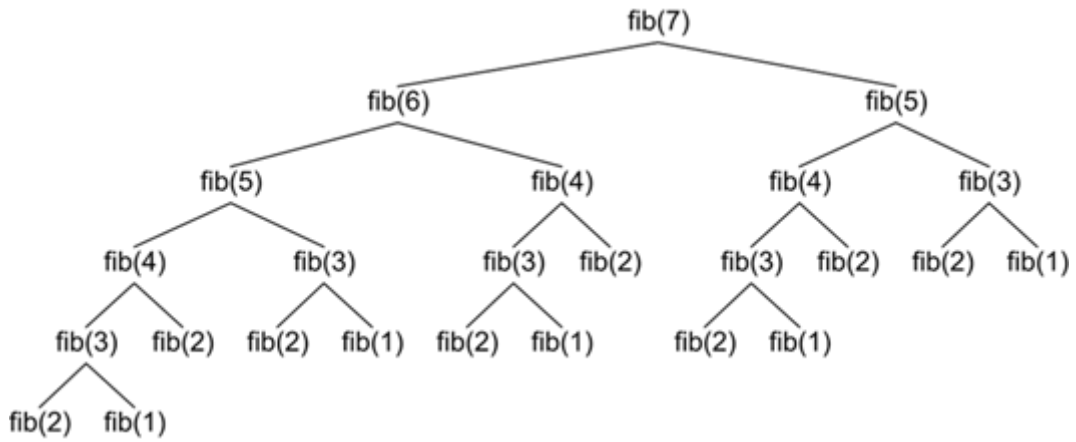
- The output should be the n^{th} Fibonacci number counting from 0

Examples

Input	Output
5	8
10	89
21	17711

Hint

For the n^{th} Fibonacci number, we calculate the N-1^{st} and the N-2^{nd} number, but for the calculation of N-1^{st} number we calculate the N-1-1^{st}(N-2^{nd}) and the N-1-2^{nd} number, so we have a lot of repeated calculations.



If you want to figure out how to skip those unnecessary calculations, you can search for a technique called memoization.

14. *Simple Text Editor

You are given an empty text. Your task is to implement **4 types of commands** related to manipulating the text:

- **"1 [string]"** - **appends** [string] to the end of the text
- **"2 [count]"** - **erases** the last [count] elements from the text
- **"3 [index]"** - **returns** the element at position [index] from the text
- **"4"** - **undoes** the last not-undone command of type 1 or 2 and returns the text to the state before that operation

Input

- The first line contains **N**, the number of operations, where **1 ≤ N ≤ 105**
- Each of the following **N** lines contains the name of the operation, followed by the command argument, if any, separated by space in the following format **"command argument"**.
- **The length of the text** will not exceed **1000000**
- All input characters are **English letters**
- It is **guaranteed** that the sequence of **input operation is possible to perform**

Output

- For each operation of type **"3"** print a **single line with the returned character of that operation**.

Examples

Input	Output	Comments
8 1 abc 3 3 2 3 1 xy 3 2 4 4 3 1	c y a	There are **8 operations**. Initially, the **text is empty**. Append "abc" Print third character Erase 3 characters Append "xy" Print second character Undo last command - text is now "" Undo last command - text is now "abc" Print first character

15. *Infix to Postfix

Mathematical expressions are **written in an infix notations**, for example "5 / (3 + 2)". However, this kind of notation is **not efficient for computer processing**, as you first need to evaluate the expression inside the brackets, so there is a lot of back and forth movement. A more suitable approach is to **convert it in the so-called postfix notations** (also called Reverse Polish Notation), in which the **expression is evaluated from left to right**, for example "3 2 + 5 /".

Implement an **algorithm that converts** the mathematical expression **from infix notation into a postfix notation**. Use the famous Shunting-yard algorithm.

Input

- You will **receive an expression on a single line, consisting of tokens**
- Tokens could be numbers 0-9, variables a-z, operators +, -, *, / and brackets (or)
- Each token is **separated by exactly one space**

Output

- The **output should be on a single line**, consisting of **tokens**, **separated by exactly one space**.

Examples

Input	Output
5 / (3 + 2)	5 3 2 + /
1 + 2 + 3	1 2 + 3 +
7 + 13 / (12 - 4)	7 13 12 4 - / +
(3 + x) - y	3 x + y -

16. **Poisonous Plants

You are given **N** plants in a garden. Each of these plants has been added with some amount of pesticide. After each day, if any plant has **more pesticide** than the plant at **its left**, being weaker (more GMO) than the left one, **it dies**. You are given the initial values of the pesticide and position of each plant. Print the number of days **after** which no plant dies, i.e. the time after which there are no plants with more pesticide content than the plant to their left.

Input

- The input consists of an integer **N** representing the number of plants.
- The next **single line** consists of **N** integers, where every integer represents the position and amount of pesticides of each plant. $1 \leq N \leq 100000$
- Pesticides amount on a plant is between 0 and 1000000000

Output

- Output a single value equal to the number of days after which no plants die

Examples

Input	Output	Comments
7 6 5 8 4 7 10 9	2	Initially all plants are alive. Plants = {(6, 1), (5, 2), (8, 3), (4, 4), (7, 5), (10, 6), (9, 7)} Plants[k] = (i, j) => j^{th} plant has pesticide amount = i. After the 1^{st} day, 4 plants remain as plants 3, 5, and 6 die. Plants = {(6, 1), (5, 2), (4, 4), (9, 7)} After the 2^{nd} day, 3 plants survive as plant 7 dies. Plants = {(6, 1), (5, 2), (4, 4)}

		After the 3rd day, 3 plants survive and no more plants die. Plants = {(6, 1), (5, 2), (4, 4)} After the 2nd day the plants stop dying.