# Exercise: Arrays

Problems for exercises and homework for the [“Technology Fundamentals” course @ SoftUni](https://softuni.bg/trainings/3212/java-fundamentals-january-2021).

You can check your solutions in [Judge](https://judge.softuni.bg/Contests/1247).

## Train

You will be given a count of wagons in a train **n**. On the next **n** lines, you will receive how many people are going to get on that wagon. At the end print the whole train and after that the sum of the people in the train.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Решение на задачата** |
| 3  13  24  8 | 13 24 8  45 | Четем броя на числата, които ще бъдат подадени. Създаваме масив с дължината на числата, които ще бъдат подадени. Създава се променлива инт сума. Правим for loop и четем подадените числа – numbers[i] = Integer.parseInt(scan.nextLine()); |
| 6  3  52  71  13  65  4 | 3 52 71 13 65 4  208 | Сумираме числата. Създавам foreach loop, чрез който за всяко подадено число int num : numbers (което се съдържа в масива numbers) да се принтира числото num заедно с интервал: System.out.print(num + “ ”);  Излизам от loop-а и принтирам със System.out.printf сумата, получена при събирането на всяко число в масива. |
| 1  100 | 100  100 |  |

## Common Elements

Write a program, which prints common elements in two arrays. You have to compare the elements of the second array to the elements of the first.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Решение на задачата** |
| Hey hello 2 4  10 hey 4 hello | 4 hello | Създавам два масива с имена firstArray и secondArray и ги чета от конзолата по следния начин: String[] firstArray = scan.nextLine().split(“ ”); . По този начин се split-ват (разделят) отделните символи. Ако входът в конзолата е {1 5 7 8}, то след split става [“1”, “2”, “7”, “8”]. |
| S of t un i  of i 10 un | of i un | Следващата стъпка е да създадем foreach loop, който обхожда **задължително** първо secondArray(защото трябва да ***сравним*** елементите **от втория** с елементите **от първия масив**. String word2:secondArray (тоест обходи всеки елемент Стринг word2, който се намира във втория масив. В този loop влагам и втори, за да обходим и първия масив. И вътре в двата loop – а слагаме проверка. |
| i love to code  code i love to | code i love to | Проверката if(word2.equals(word1)){  System.out.print(word1 + “ ”);  }  Принтира директно на конзолата всички еднакви String – ове. И с това задачата е завършена. |

## Zig-Zag Arrays

Write a program which creates 2 arrays. You will be given an integer **n**. On the next **n** lines, you get 2 integers. Form 2 arrays as shown below.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Решение на задачата** |
| 4  1 5  9 10  31 81  41 20 | 1 10 31 20  5 9 81 41 | Първоначално четем от конзолата n, което е по условие. След това създаваме два масива String[] first и secondArray = new String[n];  Следващата стъпка е да създадем for loop, чрез който до достигане на начално зададената стойност на n трябва да четем input – и от конзолата, които сплитваме, за да ги разделим със space помежду им. |
| 2  80 23  31 19 | 80 19  23 31 | String[] input = scan.nextLine().split(“ ”);  Назначаваме стойности String firstElement = input[0] и String secondElement = input[1]. Това е така, защото първия елемент от output-а с индекс 0 е именно input[0], а втория с индекс 1 е input[1].  Задачата решаваме със следния алгоритъм, който извежда числата едно през друго.  if((i+1) % 2 == 0){  secondArray[i] = firstElement;  firstArray[i] = secondElement; }else{  firstArray[i] = firstElement;  secondArray[i] = secondElement; }    Накрая приннтираме изхода с String.join()  System.*out*.println(String.*join*(" ", firstArray));  System.*out*.println(String.*join*(" ", secondArray));  По този начин. |

## Array Rotation

Write a program that receives an array and number of rotations you have to perform (first element goes at the end) Print the resulting array.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Решение на задачата** |
| 51 47 32 61 21  2 | 32 61 21 51 47 | Първоначално четем от конзолата със split един array, както и колко ротации трябва да бъдат направени(цяло число).  Създаваме for loop, който изразява нашите ротации. В него инициализираме нова променлива int firstNumber, на която поставяме стойността, намираща се на първото място или нулевия индекс в нашия масив. Int firstNumber = numbers[0]; |
| 32 21 61 1  4 | 32 21 61 1 | И сега е ред на ротацията. Във for loop-а влагаме друг for loop, който този път отразява index-ите в нашия масив. Въртим го до края на масива numbers.length-1.  Назначаваме стойност на numbers[index] = numbers[index+1]; Това се прави, за да може индекс 0 да стане индекс 1, индекс 1 да стане индекс 2 и т.н., за да може числото, което стои отдясно (индекс+1) да дойде на мястото на числото вляво от него(индекс). И така преместваме с едно наляво числата в нашия масив. |
| 2 4 15 31  5 | 4 15 31 2 | Излизаме от вложения цикъл и назначаваме на numbers[numbers.length-1] (тоест на последната позиция в масива) = firstNumber. Първото число стана последно.  Накрая правим foreach за да изпринтираме резултата.  For(int number:numbers){  System.out.print(number+ “ ”);  } |

## Top Integers

Write a program to find all the top integers in an array. A top integer is an integer which is **bigger** than all the elements to its right.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Решение:** |
| 1 4 3 2 | 4 3 2 | Четем array и го split-ваме. Създаваме for loop, за да обходим цикъла, въртим го до numbers.length-1. Назначаваме стойност на променливата int number = numbers[i], за да присвоява стойностите, зададени от потребителя.  Правим проверка дали индексът е равен на numbers.length-1. Това проверява дали сме на последното число в нашия array. Ако е така, принтираме number и break – ваме цикъла. |
| 14 24 3 19 15 17 | 24 19 17 | Във for loop – a инициализираме булева променлива с начална стойност false (isBigger=false). Влагаме for loop във вече създадения и въртим до numbers.length, започвайки от i+1, за да преместим числото с една позиция. Правим проверка отново, този път дали number>number[j] (един вид дали лявото число е по – голямо от дясното). |
| 27 19 42 2 13 45 48 | 48 | if (number > numbers[j]) {  isBigger = true; } else {  isBigger = false;  break; }  Ако е така, назначаваме на булевата променлива, създадена по – горе стойност true, ако не – false и break – ваме. Продължаваме като излизаме от for loop – a и правим проверка if(isBigger), принтираме number + “ ”. |

## Equal Sums

Write a program that determines if there **exists an element in the array** such that the **sum of the elements on its left** is **equal** to the **sum of the elements on its right**. If there are **no elements to the left / right**, their **sum is considered to be 0**. Print the **index** that satisfies the required condition or **“no”** if there is no such index.

### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Comments** | **Решение:** |
| 1 2 3 3 | 2 | At a[2] -> left sum = 3, right sum = 3  a[0] + a[1] = a[3] | Четем си aray и split-ваме. Създаваме for loop, чрез който обхождаме индексите. В него инициализираме две променливи leftSum и rightSum, които първоначално са равни на 0. Тънкият момент е, че в този for loop, ще има още два, единия за leftSum, другия – за rightSum. |
| 1 2 | no | At a[0] -> left sum = 0, right sum = 2  At a[1] -> left sum = 1, right sum = 0  No such index exists | Този за leftSum изглежда така –  for (int indexLeft = 0; indexLeft <= index-1; indexLeft++) {  leftSum += numbers[indexLeft]; }  тук забелязваме, че indexLeft<=index-1, започва от 0 |
| 1 | 0 | At a[0] -> left sum = 0, right sum = 0 | Този за rightSum така:  for (int indexRight = index + 1; indexRight <= numbers.length - 1; indexRight++) {  rightSum += numbers[indexRight]; }  тук забелязваме, че indexLeft<=index-1, но започва от index+1 |
| 1 2 3 | no | No such index exists | Вцикъла последно правим проверка, дали двете суми са равни. Горе инициализираме булева променлива със стойност false, ако сумите са равни, тя е true. И принтираме index. |
| 10 5 5 99 3 4 2 5 1 1 4 | 3 | At a[3] -> left sum = 20, right sum = 20  a[0] + a[1] + a[2] = a[4] + a[5] + a[6] + a[7] + a[8] + a[9] + a[10] | След като излезем от for loop-a правим проверка дали !isFound. Ако няма лява и дясна сума, то принтираме на конзолата „no“. |

## Max Sequence of Equal Elements

Write a program that finds the **longest sequence of equal elements** in an array of integers. If several longest sequences exist, print the leftmost one.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Решние:** |
| 2 1 1 2 3 3 **2 2 2** 1 | 2 2 2 | Първоначално прочитаме array и го split-ваме. Създаваме for loop, с помощта на който обхождаме array – a. ВЪртим го от 0 до numbers.length-1. Назначаваме стойност int currentElement = numbers[i]. Това е елементът, който разглеждаме в момента. Освен нея създаваме и променлива int nextElement = numbers[i+1], с която разглеждаме следващия елемент в array – a. Правим проверка дали настоящият елемент, който разглеждаме съвпада със следващия в масива. Ако е така, увеличаваме currentLength с 1. Създаваме int currentLength = 1 преди цикъла. ЗАЩО currentLength има първоначална стойност 1? |
| **1 1 1** 2 3 1 3 3 | 1 1 1 | Защото в началото дължината на array – a e 1, започвайки от първия елемент. С всеки следващ еднакъв с този отдясно на него, дължината се увеличава с 1, за да намерим еднаквите числа. В този if създаваме и друг, който проверява дали currentLength>bestLength. Int bestLength е създадена преди цикъла и в началото е 0. АКо е изпълнено условието, то bestLength = currentLength, a digit = currentElement. |
| **4 4 4 4** | 4 4 4 4 | Тоест digit -репрезентацията на най – дългата поредица е числото, което се повтаря. На първата проверка вторият случай – else: currentLength = 1; Тоест ако currentElement не е равен на nextElement, започва от 1 да брои, защото не е намерена достатъчна дължина. Примерно ако започва 2 4 2 2 2. Минава през 2, сравнява с 4 и вижда, че не съвпадат, означава, че започва отново да брои. Стига до 2 2 2 и ги преброява и трите. |
| 0 **1 1** 5 2 2 6 3 3 | 1 1 | Намира, че currentLength е повече от bestLength и го запазва като стойност, защото това е второто условие във вложения if.  Накрая правим for, започващ от i = 0, върти се докато i < bestLength. Принтираме digit + “ ”, числото с интервал толкова пъти че i < от bestLength, чиято стойност по – горе запазихме. От 0 в началото тя вече е запазила най – дългата поредица от числа. |

## Magic Sum

Write a program, which prints all unique pairs in an array of integers whose sum is equal to a given number.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 7 6 2 19 23  8 | 1 7  6 2 |
| 14 20 60 13 7 19 8  27 | 14 13  20 7  19 8 |

## \*Kamino Factory

The clone factory in Kamino got another order to clone troops. But this time you are tasked to find **the best DNA** sequence to use in the production.

You will receive the **DNA length** and until you receive the command **"Clone them!"** you will be receiving a **DNA sequences of ones and zeroes, split by "!"** **(one or several).**

You should select the sequence with the **longest subsequence of ones**. If there are several sequences with **same length of** **subsequence of ones**, print the one with the **leftmost** **starting index**, if there are several sequences with same **length and starting index**, select the sequence with the **greater sum** of its elements.

After you receive the last command "Clone them!" you should print the collected information in the following format:

"Best DNA sample {bestSequenceIndex} with sum: {bestSequenceSum}."

"{DNA sequence, joined by space}"

### Input / Constraints

* The **first line** holds the **length** of the **sequences** – **integer in range [1…100];**
* On the next lines until you receive **"Clone them!"** you will be receiving sequences (at least one) of ones and zeroes, **split by "!"** (one or several).

### Output

The output should be printed on the console and consists of two lines in the following format:

"Best DNA sample {bestSequenceIndex} with sum: {bestSequenceSum}."

"{DNA sequence, joined by space}"

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5  1!0!**1!1**!0  0!**1!1**!0!0  Clone them! | Best DNA sample 2 with sum: 2.  0 1 1 0 0 | We receive 2 sequences with **same length** **of subsequence of ones**, but the second is printed, because its subsequence starts at **index[1].** |
| **Input** | **Output** | **Comments** |
| 4  **1!1**!0!**1**  1!0!0!1  **1!1**!0!0  Clone them! | Best DNA sample 1 with sum: 3.  1 1 0 1 | We receive 3 sequences. Both 1 and 3 **have same length** of subsequence of ones -> 2, **and both start from index[0]**, but the first is printed, because its **sum is greater.** |

## \*LadyBugs

You are **given a field size** and the **indexes of ladybugs** inside the field. After that on every new line **until the "end" command** is given, a **ladybug changes its position** either to its **left or to its right** **by a given fly length**.

A **command to a ladybug** looks like this: "**0 right 1**". This means that the little insect placed on index 0 should fly one index to its right. If the ladybug **lands on a fellow ladybug**, it **continues to fly** in the same direction **by the same fly length**. If the ladybug **flies out of the field, it is gone**.

For example, imagine you are given a field with size 3 and ladybugs on indexes 0 and 1. If the ladybug on index 0 needs to fly to its right by the length of 1 (0 right 1) it will attempt to land on index 1 but as there is another ladybug there it will continue further to the right by additional length of 1, landing on index 2. After that, if the same ladybug needs to fly to its right by the length of 1 (2 right 1), it will land somewhere outside of the field, so it flies away:



If you are given ladybug index that does not have ladybug there, nothing happens. If you are given ladybug index that is outside the field, nothing happens.

Your job is to create the program, simulating the ladybugs flying around doing nothing. At the end, **print all cells in the field separated by blank spaces**. For each cell that has a ladybug on it print '**1**' and for each empty cells print '**0**'. For the example above, the output should be **'0 1 0'**.

### Input

* On the first line you will receive an integer - the size of the field
* On the second line you will receive the initial **indexes** of all ladybugs separated by a blank space. **The given indexes** may or may not be inside the field range
* On the next lines, until you get the "end" command you will receive commands in the format: "**{ladybug index} {direction} {fly length}**"

### Output

* Print the **all cells within the field in format: "{cell} {cell} … {cell}"**
  + If a cell has ladybug in it, print **'1'**
  + If a cell is empty, print **'0'**

### Constrains

* The size of the field will be in the range [0 … 1000]
* The ladybug indexes will be in the range [-2,147,483,647… 2,147,483,647]
* The number of commands will be in the range [0 … 100]
* The fly length will be in the range [-2,147,483,647… 2,147,483,647]

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  0 1  0 right 1  2 right 1  end | 0 1 0 | 1 1 0 - Initial field  0 1 1 - field after "0 right 1"  0 1 0 - field after "2 right 1" |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  0 1 2  0 right 1  1 right 1  2 right 1  end | 0 0 0 |
| 5  3  3 left 2  1 left -2  end | 0 0 0 1 0 |