**Exercises: Arrays**

You can check your solutions here: [https://judge.softuni.bg/Contests/Compete/Index/867#0](https://judge.softuni.bg/Contests/Compete/Index/867).

* **Day of Week**

Enter a **day number** [1…7] and print the **day name** (in English) or “**Invalid Day!**”. Use an **array of strings**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 | Monday |
| 2 | Tuesday |
| 7 | Sunday |
| 0 | Invalid Day! |

**Hints**

* Use an **array of strings** holding the day names: {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}.
* Print the element at index (**day-1**) when it is in the range [1…7] or “**Invalid Day!**” otherwise.
* **Reverse an Array of Integers**

Write a program to read **an array of integers**, **reverse** it and **print** its elements. The input consists of a **number** **n** (the number of elements) + **n** integers, each as a separate line. Print the output on a single line (space separated).

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **3**  10  20  30 | 30 20 10 |
| **4**  -1  20  99  5 | 5 99 20 -1 |

**Hints**

* First, read the number **n**.
* Allocate an array of **n** integers.
* Read the integers in a **for**-loop.
* Instead of reversing the array, you can just pass through the elements from the last (**n-1**) to the first (**0**) with a reverse **for**-loop.
* **Triple Sum**

Write a program to read **an array of integers** and find all triples of elements **a**, **b** and **c**, such that **a** + **b** == **c** (where **a** stays to the left from **b**). Print “**No**” if no such triples exist.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 1 1 1 | No |
| 4 2 8 6 | 4 + 2 == 6  2 + 6 == 8 |
| 2 7 5 0 | 2 + 5 == 7  2 + 0 == 2  7 + 0 == 7  5 + 0 == 5 |
| 3 1 5 6 1 2 | 3 + 2 == 5  1 + 5 == 6  1 + 1 == 2  1 + 2 == 3  5 + 1 == 6  1 + 2 == 3 |

**Hints:**

* Read the input numbers in array **arr[]**.
* Use nested loops to generate all pairs {**a**, **b**}, such that **0** ≤ **a** < **b** < **n**.
* Check whether **arr[]** contains the sum **arr[a]** **+** **arr[b]**.
* **Sum Arrays**

Write a program that reads two **arrays of integers** and sums them. When the arrays are not of the same size, duplicate the smaller array a few times.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 2 3 4  2 3 4 5 | 3 5 7 9 | 1 2 3 4 +  2 3 4 5 =  3 5 7 9 |
| 1 2 3 4 5  2 3 | 3 5 5 7 7 | 1 2 3 4 5 +  2 3 2 3 2 =  3 5 5 7 7 |
| 5 4 3  2 3 1 4 | 7 7 4 9 | 5 4 3 5 +  2 3 1 4 +  7 7 4 9 |

**Hints**

* Assume the first array **arr1** has **len1** elements and the second **arr2** has **len2** elements.
* The result array will have **max**(**len1**, **len2**) elements.
* We sum array elements one by one (from the first to the last). To enable **rotating** (take the first element as next after the last), we use the **position** **%** **length** indexing: **arr1[i** **%** **len1]** and **arr2[i** **%** **len2]**.
* **Condense Array to Number**

Write a program to read **an array of integers** and **condense** them by **summing** adjacent couples of elements until a **single integer** is obtained. For example, if we have 3 elements {2, 10, 3}, we sum the first two and the second two elements and obtain {2+10, 10+3} = {12, 13}, then we sum again all adjacent elements and obtain {12+13} = {25}.

**Examples**

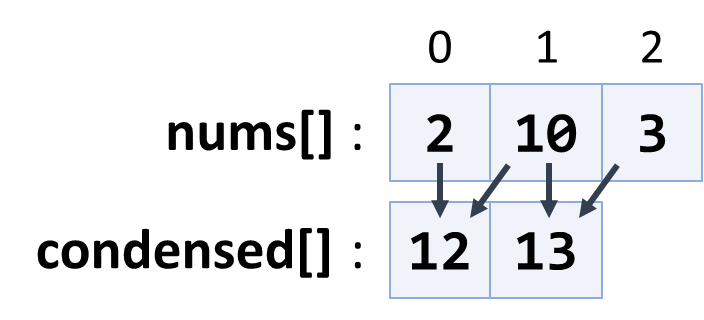
|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2 10 3 | 25 | 2 10 3 2+10 10+3 12 13 12 + 13 25 |
| 5 0 4 1 2 | 35 | 5 0 4 1 2 5+0 0+4 4+1 1+2 5 4 5 3 5+4 4+5 5+3 9 9 8 9+9 9+8 18 17 18+17 35 |
| 1 | 1 | 1 is already condensed to number |

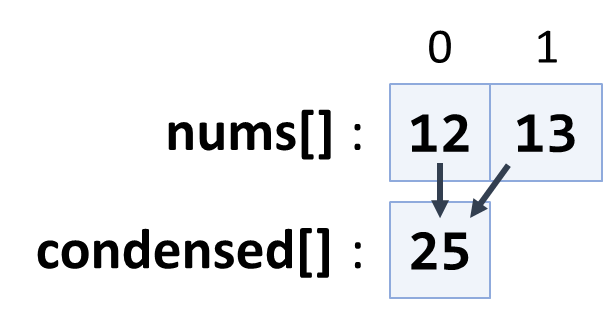
**Hints**

While we have more than one element in the array **nums[]**, repeat the following:

* Allocate a new array **condensed[]** of size **nums.Length-1**.
* Sum the numbers from **nums[]** to **condensed[]**:
* **condensed[i] = nums[i] + nums[i+1]**
* **nums[] = condensed[]**

The process is illustrated below:





* **Extract Middle 1, 2 or 3 Elements**

Write a method to extract the **middle** **1**, **2** or **3** **elements** from array of **n** integers and **print** them.

* **n** = 1 -> **1** element
* even **n** -> **2** elements
* odd **n** -> **3** elements

Create a program that reads an **array of integers** (space separated values) and prints the middle elements in the format shown in the examples.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| **5** | { 5 } |
| 2 3 **8 1** 7 4 | { 8, 1 } |
| 1 2 **3 4 5** 6 7 | { 3, 4, 5 } |
| 10 20 30 **40 50** 60 70 80 | { 40, 50 } |

**Hints**

* Write different logic for each case (n = 1, even n, odd n)
* n = 1 take the first element
* odd n take elements n/2-1, n/2, n/2+1
* even n take elements n/2-1 and n/2
* **Last K Numbers Sums Sequence**

Enter two integers **n** and **k**. Generate and print the following sequence of **n** elements:

* The first element is: **1**
* All other elements = sum of the previous **k** elements (if less than **k** are available, sum all of them)
* Example: n = **9**, k = **5** **120** = 1 + 1 + 2 + 4 + 8 + 16 + 31 + 61

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  3 | 1 1 2 4 7 13 |
| 8  2 | 1 1 2 3 5 8 13 21 |
| 9  5 | 1 1 2 4 8 16 31 61 120 |

**Hints**

* Use an **array of integers** to hold the sequence.
* Initially **seq[0] = 1**
* Use two nested loops:
* Loop through all elements **i** = **1 … n**
* Sum the elements **i-k … i-1**: **seq[i] = sum(seq[i-k … i-1])**
* **Largest Common End**

Read **two** **arrays** **of** **words** and find the length of the **largest common end** (left or right).

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| **hi php java** csharp sql html css js  **hi php java** js softuni nakov java learn | 3 | The largest common end is at the left: **hi php java** |
| hi php java xml csharp **sql html css js**  nakov java **sql html css js** | 4 | The largest common end is at the right: **sql html css js** |
| I love programming  Learn Java or C# | 0 | No common words at the left and right |

**Hints**

* Scan the arrays from left to right until the end of the shorter is reached and count the equal elements.
* Scan the arrays form right to left until the start of the shorter is reached.
* Keep the start position and the length of the longest equal start / end.
* **Rotate and Sum**

To “**rotate** an array on the right” means to move its last element first: {1, 2, 3} {3, 1, 2}.

Write a program to read an array of **n** **integers** (space separated on a single line) and an integer **k**, rotate the array right **k** **times** and sum the obtained arrays after each rotation as shown below.

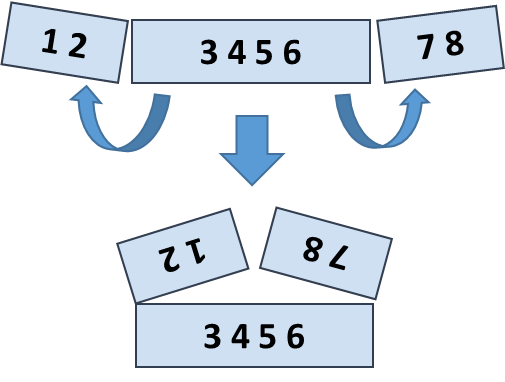
**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3 2 4 -1  2 | 3 2 5 6 | rotated1[] = -1 3 2 4  rotated2[] = 4 -1 3 2  sum[] = 3 2 5 6 |
| 1 2 3  1 | 4 3 5 | rotated1[] = 3 1 2  sum[] = 4 3 5 |
| 1 2 3 4 5  3 | 12 10 8 6 9 | rotated1[] = 5 1 2 3 4  rotated2[] = 4 5 1 2 3  rotated3[] = 3 4 5 1 2  sum[] = 12 10 8 6 9 |

**Hints**

* After **r** rotations the element at position **i** goes to position **(i + r) % n**.
* The **sum[]** array can be calculated by two nested loops: for **r** = **1** … **k**; for **i** = **0** … **n-1**.
* **Fold and Sum**

Read an array of **4\*k** integers, fold it like shown below, and print the sum of the upper and lower two rows (each holding 2 \* k integers):



**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5 **2 3** 6 | 7 9 | 5 6 +  2 3 =  7 9 |
| 1 2 **3 4 5 6** 7 8 | 5 5 13 13 | 2 1 8 7 +  3 4 5 6 =  5 5 13 13 |
| 4 3 -1 **2 5 0 1 9 8**  6 7 -2 | 1 8 4 -1 16 14 | -1 3 4 -2 7 6 +  2 5 0 1 9 8 =  1 8 4 -1 16 14 |

**Hints**

* Create the **first row** after folding: the first **k** numbers reversed, followed by the last **k** numbers reversed.
* Create the **second row** after folding: the middle 2\***k** numbers.
* **Sum** the first and the second rows.
* **Compare Char Arrays**

Compare two char arrays lexicographically (letter by letter).

Print the them in alphabetical order, each on separate line.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| a b c  d e f | abc  def |
| p e t e r  a n n i e | annie  peter |
| a n n i e  a n | an  annie |
| a b  a b | ab  ab |

**Hints**

* Compare the first letter of **arr1[]** and **arr2[]**, if equal, compare the next letter, etc.
* If all letters are equal, the smaller array is the **shorter**.
* If all letters are equal and the array lengths are the same, the arrays are **equal**.
* **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** |
| 2 1 1 2 3 3 **2 2 2** 1 | 2 2 2 |
| **1 1 1** 2 3 1 3 3 | 1 1 1 |
| **4 4 4 4** | 4 4 4 4 |
| 0 **1 1** 5 2 2 6 3 3 | 1 1 |

**Hints**

* Start with the sequence that consists of the first element: **start**=**0**, **len**=**1**.
* Scan the elements from left to right, starting at the second element: **pos**=**1**…**n-1**.
* At each step compare the current element with the element on the left.
* Same value you have found a sequence longer by one **len**++.
* Different value start a new sequence from the current element: **start**=**pos**, **len**=**1**.
* After each step remember the sequence it is found to be longest at the moment: **bestStart**=**start**, **bestLen**=**len**.
* Finally, print the longest sequence by using **bestStart** and **bestLen**.
* **Max Sequence of Increasing Elements**

Write a program that finds the **longest increasing subsequence** in an array of integers. The longest increasing subsequence is a **portion of the array** (subsequence) that is strongly **increasing** and has the **longest possible length**. If several such subsequences exist, find the left most of them.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 **2 3 4** 2 2 4 | 2 3 4 |
| 4 5 **1 2 3 4 5** | 1 2 3 4 5 |
| **3 4 5 6** | 3 4 5 6 |
| **0 1** 1 2 2 3 3 | 0 1 |

**Hints**

* Use the same algorithm like in the previous problem (Max Sequence of Equal Elements).
* **Most Frequent Number**

Write a program that finds the **most frequent number** in a given sequence of numbers.

* Numbers will be in the range [0…65535].
* In case of multiple numbers with the same maximal frequency, print the left most of them.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Output** |
| **4** 1 1 **4** 2 3 **4 4** 1 2 **4** 9 3 | 4 | The number **4** is the most frequent (occurs 5 times) |
| **2 2 2 2** 1 **2 2 2** | 2 | The number **2** is the most frequent (occurs 7 times) |
| **7 7 7** 0 2 2 2 0 10 10 10 | 7 | The numbers **2**, **7** and **10** have the same maximal frequence (each occurs 3 times). The leftmost of them is **7**. |

* **Index of Letters**

Write a program that creates an array containing all letters from the alphabet (**a**-**z**). Read a lowercase word from the console and print the **index of each of its letters in the letters array**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| abcz | a -> 0  b -> 1  c -> 2  z -> 25 |
| softuni | s -> 18  o -> 14  f -> 5  t -> 19  u -> 20  n -> 13  i -> 8 |

* **Pairs by Difference**

Write a program that **count the number of pairs** in given array **which** **difference is equal to given number**.

**Input**

* The **first line** holds the **sequence of numbers**.
* The **second line** holds the **difference**.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 5 3 4 2  2 | 3 | Pairs of elements with difference 2 -> {1, 3}, {5, 3}, {4, 2} |
| 5 3 8 10 12 1  1 | 0 | No pairs with difference 1 |

* **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** |
| 1 2 3 3 | 2 | At a[2] -> left sum = 3, right sum = 3  a[0] + a[1] = a[3] |
| 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 |
| 1 | 0 | At a[0] -> left sum = 0, right sum = 0 |
| 1 2 3 | no | No such index exists |
| 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] |