# Exercises: Arrays

Problems for exercises and homework for the [“Programming Fundamentals” course @ SoftUni](https://softuni.bg/courses/programming-fundamentals).

You can check your solutions here: <https://judge.softuni.bg/Contests/172/Arrays>.

## 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.

## Sum, Min, Max, First, Last, Average

Write a program to read **n** integers and print their **sum**, **min**, **max**, **first**, **last** and **average** values.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **5**  12  20  -5  37  8 | Sum = 72  Min = -5  Max = 37  First = 12  Last = 8  Average = 14.4 |
| **4**  50  20  25  40 | Sum = 135  Min = 20  Max = 50  First = 50  Last = 40  Average = 33.75 |

### Hints

* Include the “System.Linq” namespace to enable aggregate functions.
* Read the input array nums[].
* Use nums.Min(), nums.Max(), nums.First(), etc.

## 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** = 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**])

## 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 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 + 5 == 5  1 + 5 == 6  1 + 1 == 2  1 + 2 == 3  5 + 1 == 6  1 + 2 == 3 |

### Hints:

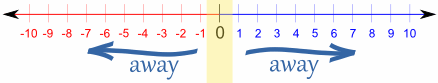
* Read the input numbers in array arr[].
* Use nested loops to generate all pairs {a, b}, such that 0 ≤ a < b < n.
* Check whether arr[] contains the sum arr[a] + arr[b].

## Rounding Numbers Away from Zero

Write a program to read **an array of real numbers** (space separated values), **round** them to the nearest integer in “**away from 0**” style and **print** the output as in the examples below.

Rounding in “[away from zero](https://www.mathsisfun.com/numbers/rounding-methods.html)” style means:

* To round to the nearest integer, e.g. 2.9 🡪 3; -1.75 🡪 -2
* In case the number is exactly in the middle of two integers (midpoint value), round it to the next integer which is away from the 0:



### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 0.9 1.5 2.4 2.5 3.14 | 0.9 => 1  1.5 => 2  2.4 => 2  2.5 => 3  3.14 => 3 |
| -5.01 -1.599 -2.5 -1.50 0 | -5.01 => -5  -1.599 => -2  -2.5 => -3  -1.50 => -2  0 => 0 |

### Hints:

* **Variant I**: Take the **absolute value** of each input number, add **0.5** and **truncate** the integral part. If the original number is negative, make the result also negative.
* **Variant II**: **Search in Internet** for **“rounding away from zero” + C#**. You should find a build-in C# method for rounding in many styles. Choose “away from zero” rounding.

## Reverse an Array of Strings

Write a program to read **an array of strings**, **reverse** it and **print** its elements. The input consists of a sequence of space separated strings. Print the output on a single line (space separated).

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| a b c d e | e d c b a |
| -1 hi ho w | w ho hi -1 |

### Hints

* Read the array of strings.
* **Exchange** the **first** element (at index 0) with the **last** element (at index n-1).
* **Exchange** the **second** element (at index 1) with the element **before the** **last** (at index n-2).
* Continue the same way until the middle of the array is reached.



* Another, shorter approach, is to use the .Reverse() extension method from “System.Linq”.

## 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