# Passing an array to a function

**The goal for this exercise** is to make sure that you pass an array, as a parameter, to a method. Also: to give you more practice creating and using methods/functions.

In the starter project, you need to add code to the Array\_Parameter class, so that the RunExercise method does the following:

* 1. Create the array in RunExercise, and name it something like dataValues.
	2. Call the GetUserData method, which will fill in the array by asking the user for a value for each space in the array.
	3. Call the CalculateSum method, which will get the sum of the non-negative values in the array, and return the sum to RunExercise.
	4. RunExercise will store that sum into a variable
	5. RunExercise will print the sum (using the variable)

Here is some more information on the methods you'll write:

* **GetUserData**

It returns an integer value representing the number of integers entered by the user, and accepts the following arguments:

* + values - an array of integers that will be used to store the data entered by the user

It allows the user to enter up to values.Length positive integers at the keyboard, and stores them in consecutive elements of the array values. If the user wants to enter fewer than values.Length positive integers, they can enter a negative integer as a sentinel indicating that they are finished entering data. This negative integer should NOT be entered into the array. *This method should return the total number of values actually stored in the array.*

* **CalculateSum**

It returns the sum of the non-negative integers stored in an array, and accepts the following arguments:

* + howMany- a SINGLE integer, that tells the method how many array slots to (potentially) include in the sum. CalculateSum always starts including the 0th element, and works it’s way up. So if this parameter is 3, then the numbers stored in values[0], values[1], and values[2] will be used, assuming that they’re all non-negative.
		- Note that if howMany is larger than the length of the values array, your method MUST NOT CRASH! Thus, you will need to make sure that your code doesn’t accidentally access anything beyond the bounds of the array if howMany is too large.
	+ values - an array of integers (note that this parameter isn't listed with the method's name, above)

You should write your code so that the uncommented code can generate output to the user such as the following. User input is in **bold**; note that in the second example, when the array of 10 elements is completely filled up, the GetUserData method stops asking for input.

First example:

Please type in a number. Type a negative number to exit!

**3**

Please type in a number. Type a negative number to exit!

**4**

Please type in a number. Type a negative number to exit!

**-1**

Stored 2 values into the array

The total is: 7

Second example:

Please type in a number. Type a negative number to exit!

**1**

Please type in a number. Type a negative number to exit!

**1**

Please type in a number. Type a negative number to exit!

**1**

Please type in a number. Type a negative number to exit!

**1**

Please type in a number. Type a negative number to exit!

**1**

Please type in a number. Type a negative number to exit!

**1**

Please type in a number. Type a negative number to exit!

**1**

Please type in a number. Type a negative number to exit!

**1**

Please type in a number. Type a negative number to exit!

**1**

Please type in a number. Type a negative number to exit!

**1**

Stored 10 values into the array

The total is: 10

**What you need to do for this exercise:**

1. In the starter project, you need to implement the GetUserData and the CalculateSum methods in the Array\_Parameter class, so that they function as described above.

	1. There is a block of commented-out code for the Array\_Parameter class, including some ‘sample code’ that helps to demonstrate how to use the methods that you’ll be writing. You need to uncomment this, and build your answer on it

		1. You can quickly (un)comment a lot of code by selecting the code, then using the menu item Edit 🡪 Advanced 🡪 (Un)Comment Selection
	2. These methods will be tested mostly based on the return value handed back by the methods, combined with the test verifying that the array argument contains the correct values once the method has finished.

		1. Because of this, you’re free to have the messages that GetUserData prints vary from what’s listed above.
		2. You are NOT free to change the order in which the method accepts user input. In a nutshell, the test will ‘preload’ fake “user input” to the method, and then call GetUserData, so the method HAS to accept a series of numbers (until it either runs out of space in the array, or else until it encounters a negative number).
		3. GetUserData may assume that it will only be given integer values. It does not need to check for non-integer numbers, nor does it need to check for errors like the user typing in their name instead of a number, etc.
			1. You’re welcome to add input-validation for this, you’re just not required to ☺
	3. The methods need to handle arrays that are zero-length, and ‘bad’ inputs for integer parameters (i.e., zero and negative numbers for the ‘howMany’ parameter of CalculateSum – the sum should be zero).
2. You need to uncomment and then run the tests in the NUnit\_Tests\_Passing\_Arrays\_To\_Functions class, and make sure that all the tests pass.

	1. Note that you may see IndexOutOfBounds errors in your test – what this means is that the test has asked your code to do something, and your code went beyond the bounds of the array. This might be a negative index, or (more likely) your code is using indices that are too big for the array.

		1. I'd recommend putting in a lot of Console.WriteLine statements to try and determine where the error(s) is (are). Printing out something before each step of a loop is a good way to go.