# Passing an array to a function

**The goal for this exercise** is to make sure that you can pass an array, as a parameter, to a method. Also: to give you more practice creating an 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.
     1. Notice that there's a bunch of code provided to you already, and much of it is commented out. You'll need to uncomment that code, then FIX that code, because the code is there to provide guidance on what you should be doing (and NOT to simply give you an example solution)
  2. Call the GetUserData method (in the ArrayHelper class), which will fill in the array by asking the user for a value for each space in the array.
  3. Call the CalculateSum method (in the ArrayHelper class), which will get the sum of the positive values in the array that are less than 10, and return the sum to RunExercise.
     1. Make sure that you use the .Length property of the array parameter, so that this method can work with any sized array (and not just the one particular array you're using for this exercise)
     2. Example: If the array contains {-1, 2, 0, 20, 3} then CalculateSum should produce 2 + 3 = 5. -1 and 0 are not positive (technically positive numbers are > 0 ), and 20 is higher than 10 and therefore also excluded.
  4. RunExercise will print the sum (if you want to, you can store that sum into a variable before printing it)

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 method should return the total number of values actually stored in the array.*

* **CalculateSum**

It returns the sum of the positive integers that are less than 10 that are stored in an array (as described below), 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 only include the positive numbers in your sum. You should do this even though the GetUserData will not input negative numbers into the array. Remember that CalculateSum is a public method, which anyone, anwhere can call. Even if you’re not currently calling it with negative values in the array you could easily do so in the future, so make sure that this method only totals up the positive numbers in the array.
* Even these two methods are part of the same program, and will be used one right after the other, you should write the above two methods as if they were two independent, public methods that anyone could call at any time.   
  In a way, this is what the ‘public’ marker on the methods really means – anyone, anywhere, can call these methods from any class. As such, all public methods really ought to safeguard themselves from things like bad parameters all the time. Frequently we omit this, but such safeguards really are best practice.   
  In particular, the CalculateSum shouldn’t assume anything about the array that it’s given (nor about the howMany parameter). Make sure that CalculateSum will correctly calculate the sum of the first howMany the positive values in the array, and that it won’t crash if howMany is greater than the length of the array.

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.

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

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 ArrayHelper 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.   
      Some of this code you will have to modify to complete this exercise, and the rest of it you are allowed to modify (but don't modify the code to the point where it avoids the goal of this exercise ☺ )
   2. 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). They must handle these values even if it is not possible for your particular program to generate these bad values.   
      Remember that you’re defining a public method, and it’s possible for someone else to reuse these methods in situations where the bad inputs might occur, so you must check for (and deal with!) these errors.
   3. While it's recommended that you follow the above pattern for the user input and output, you're not required to match the output exactly. Anything that's pretty close to what's listed above will be fine.