OVERVIEW

Most of the data we encounter in everyday life occurs in groups rather than in individual pieces. We tend to collect these related data items into conceptual units to which we assign names—a group of birds is a flock, a collection of students is a class, and so forth. In C#, arrays are used to store groups of similar items in a named collection. What makes array particular useful is the ease and efficiency with which you can access individual items, particularly when all (or some) of the items in the array are accessed as part of an iterative process—when finding the largest bird in a flock, for instance.

BACKGROUND

An **array** is a named collection of data items. All the items in an array must be of the same type. You can create arrays of integers, array of strings, and so forth. An array declaration specifies the name of the array, the type of elements stored in the array, and the number of array elements (the size of the array). In the following program the array `pollutionLevel` is used to store a set of pollution level readings.

```
// Reads six integers into array pollutionLevel and displays them
// to the screen three per line
class Program
{
    static void Main(string[] args)
    {
        const int POLLUTION_ARRAY_SIZE = 6;
        // Array of six pollution level readings
        int[] pollutionLevel = new int[POLLUTION_ARRAY_SIZE];
        string userInput;   // To hold user keyboard input value


        // Input six integers into the pollution level array.
        Console.WriteLine("Enter the six pollution level readings: ");
        for( int index = 0; index < POLLUTION_ARRAY_SIZE; index++ )
        {
            userInput = Console.ReadLine();
            pollutionLevel[ index ] = int.Parse( userInput );
        }

        // Display the readings three values per line.
        for( int index = 0; index < POLLUTION_ARRAY_SIZE; index++ )
        {
            if (index % 3 == 0)
            {
                Console.WriteLine();
            }
            Console.Write($"{pollutionLevel[index]}");
        }
        Console.WriteLine();
    }
}
```

The declaration

```
int[] pollutionLevel = new int[POLLUTION_ARRAY_SIZE];
```

creates an array of integers named `pollutionLevel` and reserves enough memory to store six integers.

The elements in an array are numbered beginning with zero. You refer to an individual array element by placing its number—called its **subscript** or **array index**—within square brackets immediately after the array name. You denote the first element in the `pollutionLevel` array as `pollutionLevel[0]` the second as `pollutionLevel[1]`, and so forth.

In the preceding program, a series of pollution level readings are input using a loop in which the loop counter, index, ranges from 0 to 5. For each value of index, the statement

```
pollutionLevel[ index ] = int.Parse( Console.ReadLine() );
```

reads in a pollution level and stores it in array element `pollutionLevel[index]`. For example if you entered the integer values

```
40
25
15
12
31
43
```

as the six pollution-level readings, your input data would be stored in `pollutionLevel` as followings:

```
pollutionLevel[0]    40
pollutionLevel[1]    25
pollutionLevel[2]    15
pollutionLevel[3]    12
pollutionLevel[4]    31
pollutionLevel[5]    43
```

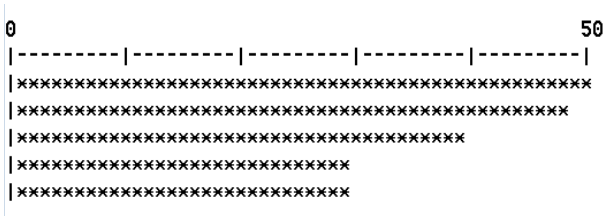
WARM-UP EXERCISE

Complete the following program by filling in the missing C# code.

```
// Finds the average of up to 100 scores.
class Program
{
    static void Main(string[] args)
    {
        const int MAX_NUM_SCORES = ______________;            // Max number of scores

        int count;                                              // Actual number of scores
        double[] score = new double[ _____________________________ ];    // Array of size
        double sum = 0;                         // Sum of array elements
        string userInput;                // To hold user keyboard input value

        // Prompt the user for the number of scores.
        Console.WriteLine("Enter the number of scores: ");
        userInput = Console.ReadLine();
        count = int.Parse(userInput);

        // Read in the scores and store them in the array.
        Console.WriteLine("Enter the scores: ");
        for( int index = _____; index < _______________; index++ )
        {
            userInput = Console.ReadLine();
            ______________________________ = double.Parse(userInput);
        }
        Console.WriteLine();

        // Find and display the average of the scores
        for( int index = ____; index < ______________; index++ )
        {
            _____________________________________________________________;    // Sum the scores
        }
        Console.WriteLine($"The average is {sum / count}",);
    }
}
```

A PICTURE'S WORTH A THOUSAND VALUES

Data is often easier to interpret when it is displayed in a graphic form. Pie charts, line graphs, and bar charts are all examples of graphic representation of data. In this exercise you generate bar graphs like the one displayed below:



```
0                                                50
|---------|---------|---------|---------|---------|
|**************************************************
|************************************************
|***************************************
|*****************************
|*****************************
```

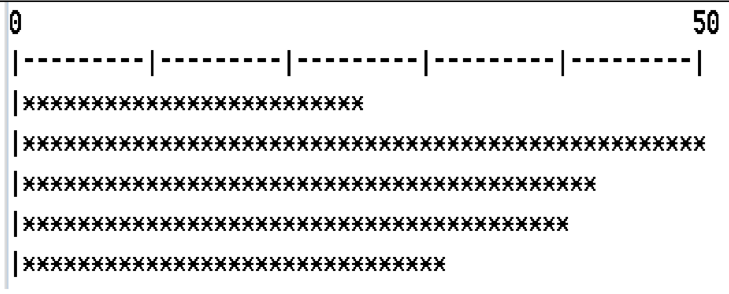
Step 1: Create a program named ***Bar1*** that displays a bar graph from a set of integer values entered by the user. The user first enters the number of bars in the graph, followed by a set of integer values (one per bar). Your program then displays a bar graph in which each bar is formed using a row of asterisk. From example, a row representing the integer value 38 would contain 38 asterisk. Assume that the graph can have no more than 10 bars and that the integer values range from 0 to 50.

Input:	The number of bars (up to 10)
	A list of integer values in the range 0 to 50
Output:	A bar graph

Be sure to display the bar graph scale (0-50) shown in the preceding bar graph.

Step 2: Complete the following test plan.

Test Plan for *Bar1*			
Test case	*Test data*	*Expected result*	*Checked*

Bar graph with the max data value	5
	25
	50
	42
	40
	31
		
		`0                                                50`
		`|---------|---------|---------|---------|---------|`
		`|*************************`
		`|**************************************************`
		`|******************************************`
		`|****************************************`
		`|*******************************`	

Bar graph with the max number of bars	10		