BACKGROUND

When you call a method with an integer argument, a copy of the argument is created, and this copy is passed to the method. Passing arrays in this way is quite costly in terms of both the time it takes to create the copy and the memory required to store the copy. For the sake of efficiency, C# uses a different mechanism, called **pass-by-reference**, to pass array arguments to methods.

When an array argument is passed to a method, the method is given the address in memory where the first element in the array is stored. As a result, changes made to the corresponding array parameter change the array argument also. The following program passes an array argument to the method that modifies the contents of the array.

```
// Passes an array to a method and doubles the values in the array
class Program
{
    const int MAX_NUM_VALUES = 20;      // Max number of data values
    const int MAX_PER_LINE = 5;         // Max values per line to display
    static void Main(string[] args)
    {
        int numValues;                          // Actual number of values to process
        int[] value = new int[MAX_NUM_VALUES];  // Array of integers
        string userInput;   // To hold user keyboard input value
        // Prompt the user for the number of data items.
        Console.Write("Enter the number of values to process: ");
        userInput = Console.ReadLine();
        numValues = int.Parse(userInput);
        // Read in the data.
        InputValues(value, numValues);

        // Double the value of each data element.
        DoubleValues(value, numValues);

        // Display the data.
        Console.WriteLine("Data doubled:");
        DisplayValues(value, numValues);
        Console.WriteLine();
    }

    // ---------------------------------------------------------------------
    static void InputValues( int[] data, int count )
    {   // Read the values into an array
        Console.WriteLine("Enter the data: ");
        string userInput;   // To hold user keyboard input value
        for (int index = 0; index < count; index++ )
        {
            userInput = Console.ReadLine();
            data[index] = int.Parse(userInput);
        }
    }
```

```
// ---------------------------------------------------------------------
    static void DoubleValues( int[] data, int count )
    {   // Double the values in the array
        for (int index = 0; index < count; index++)
        {
            data[index] = data[index] * 2;
        }
    }

    // ---------------------------------------------------------------------
    static void DisplayValues( int[] data, int count )
    {   // Display the values in the array.
        for (int index = 0; index < count; index++)
        {
            if( index % MAX_PER_LINE == 0 )
            {
                Console.WriteLine();
            }
            Console.Write($"{data[index]}");
        }
    }
```

Let's look at the **`InputValues()`** method. The empty square brackets before **`data`** in the method declaration signature

```
static void InputValues( int[] data, int count )
```

signals the compiler that **`data`** is an array. The call

```
InputValues(value, numValues);
```

passes the array `value` to the parameter **`data`**. In effect, **`data`** is nothing more than an alias for **`value`**. Any changes that are made to data in **`InputValues()`** are actually made to **`value`**. The same is true for the following call:

```
static void DoubleValues( int[] data, int count )
```

The use of pass-by-reference is reflected in the following sample output:

```
Enter the number of values to process: 7
Enter the data:
34
50
43
21
39
46
44
Data doubled:

68 100 86 42 78
92 88
```

WARM-UP EXERCISE

The following specifications describe methods that read in mileage data for a series of trips, convert the distance traveled from miles to kilometers for each trip, and display the length of each trip in both miles and kilometers.

```
static void ReadMiles ( double[] tripMiles, int count  )
```
Input parameters
count: number of trips
Output parameter
tripMiles[] : number of miles in each trip (read from keyboard)

```
static void MilesToKms ( double[] tripMiles, double[] tripKms, int count )
```
Input parameters
count: number of trips
tripMiles[] : number of miles in each trip
Output parameter
tripKms[] : number of kilometers in each trip

```
static void DisplayData ( double[] tripMiles, double[] tripKms, int count   )
```
Input parameters
count: number of trips
tripMiles[] : number of miles in each trip
tripKms[] : number of kilometers in each trip
Outputs
Displays each trip in both miles and kilometers

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

```
// Displays the distance travelled for up to 100 trips in both miles
// and kilometers
class Program
{
    const int MAX_TRIPS = 100;          // Max number of trips
    const double MILES_TO_KMS = 1.61;   // Miles to kilometers conversion factor
    static void Main(string[] args)
    {
        int numTrips;                               // Actual number of trips
        double[] miles = new double[MAX_TRIPS];     // Miles data
        double[] kms = new double[MAX_TRIPS];       // Kilometers data
        string userInput;   // To hold user keyboard input value
        // Prompt the user for the number of trips.
        Console.WriteLine("Enter the number of trips: ");
        userInput = Console.ReadLine();
        numTrips = int.Parse(userInput);
        // Read the mileage for each trip.
        ReadMiles(_____________________________________________);
        // Convert from miles to kilometers.
        MilesToKms(_____________________________________________);
        // Display the data for each trip.
        DisplayData(_____________________________________________);
    }
```

```
    // ------------------------------------------------------------------
    static void ReadMiles(double[] tripMiles, int count)
    {
        string userInput;   // To hold user input value
        Console.WriteLine("Enter the mileage for each trip: ");
        for( int index = 0; index < count; index++ )
        {
            userInput = Console.ReadLine();

            ____________________________________________________________________________________;
        }

    }
    static void MilesToKms(double[] tripMiles, double[] tripKms, int count)
    {
        for (int index = 0; index < count; index++)
        {
            _____________________ = _____________________ * _____________________;
        }
    }
    static void DisplayData(double[] tripMiles, double[] tripKms, int count)
    {
        Console.WriteLine($"{"Trip",4} {"Miles",7} {"Kilometers",12}");

        for (_____________________________________________________________________________)
        {
            Console.WriteLine($"{index + 1,4}" +

                    $"{_____________________________________,7}" +

                    $"{_____________________________________,12}"
            );
        }
    }
}
```

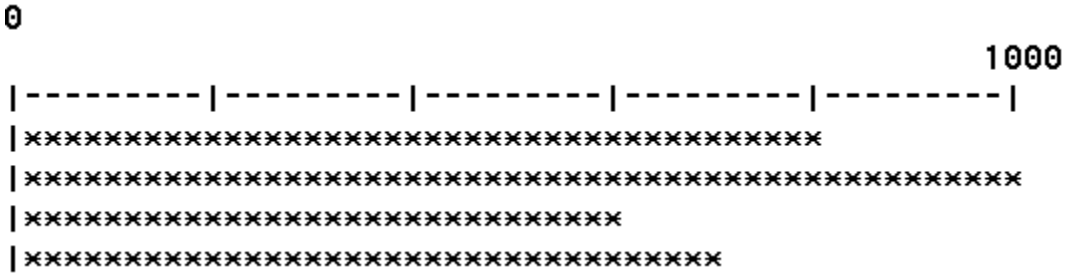
SCALED DOWN

The bar graph program you created earlier could only display integer values in the range 0 to 50. In this exercise you generalize your program to support a much broader range of values by scaling the data before displaying it. The data values are scaled to the range 0 to 50 using the formula:

$$Scaled\ value = \ data\ value * \left(\frac{50}{maximum\ data\ value}\right)$$

Scaling the values 800, 1000, 600, and 700 using this formula produces the scaled values 40, 50, 30, and 35, respectively. Note that the maximum data value (1000) yields a scaled value of 50, and the remaining values are reduced linearly in proportion to the maximum data value.

Having scaled the data, your new program will then display it. The values listed above produce the following scaled bar graph:



Step 1: Using your program ***Bar1*** as a basis, create a program named ***Bar2*** that displays a scaled bar graph. Base your new program on the method specified below:

```
static void ReadData ( double[] data, int numValues )
```
Input parameters
numValues: number of values (bars)
Output parameter
data[] : data values (read from keyboard)

```
static double MaxDataValue ( double[] data, int numValues )
```
Input parameters
data[] : data values
numValues: number of values (bars)
Returns
The largest data value

```
static void ScaleValues ( double[] data, int[] scaledData,
                          int numValues, double maxValue)
```
Input parameters
data[]: data values
numValues: number of values (bars)
maxValue: largest data value
Output parameter
scaledData[] : scaled data values

```
static void DisplayBarGraph ( int[] scaledData, int numValues, double maxValue)
```

Input parameters

`scaledData[]`: scaled data values

`numValues`: number of values (bars)

`maxValue`: largest data value

Outputs

A scaled bar graph.

Step 2: Complete the following test plan.

Test Plan for *Bar2*			
Test case	*Test data*	*Expected result*	*Checked*
Bar graph scaled by 50/100	5 80 80.8 100 73 25	0 100 |---------|---------|---------|---------|---------| |xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx |xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx |xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx |xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx |xxxxxxxxxxx	
Bar graph scaled by 50/2500			