

EXERCISE 1

The tasks in the first part of this section are step by step instructions to guide you in the practical application of the theory of programming. This is followed by a number of tasks for you to complete by applying the knowledge that you have previously gained.

TASK 1: CREATING AN ARRAY

In this task you will create an array whose element type is the data type double. The array will have 10 elements, and you should individually initialize each element. After you have initialized the elements, you should individually print each element by referencing the array.

Step 1:

Open the Visual Studio File menu, and then select **New** (or press `Ctrl+Shift+N`). Then click on **Project** and name it e.g. `Prac5Task1`.

Click on the solution name from the Solution Explorer, go to **New** and select **Class** to add a new class. This opens the New Class dialog box. Then enter a name for the new class. Use an appropriate file name e.g. `CreateArray`

Step 2: Declare the array

To use an array in a program, you must declare a variable to reference the array (the name of the array) and you must specify the type of the array variable. Here is the syntax for declaring an array variable:

```
typeName [] arrayRefVar;
```

`typeName` can refer to any of the simple data types or objects either defined by the user.

To construct the array use the new operator as follows:

```
arrayRefVar = new typeName[length];
```

The above statement does two things:

- It creates an array using `new typeName[length];`
- It assigns the reference of the newly created array to the variable `arrayRefVar`.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

```
typeName [] arrayRefVar = new typeName[length];
```

```
int [] anArray = new int [length]
```

Table 1 below outlines how you can declare arrays for different data types:

Declaring Arrays	
<pre>int[] numbers = new int[10];</pre>	An array called numbers of ten integers. All elements are initialized to zero.
<pre>final int LENGTH = 10; int[] numbers = new int[LENGTH];</pre>	It is a good idea to use a named constant (LENGTH) instead of a "magic number."
<pre>int length = in.nextInt(); double[] data = new double[length];</pre>	The length need not be a constant. It can be taken from an input stream.
<pre>int[] squares = {0, 1, 4, 9, 16};</pre>	An array of five integers with initial values.
<pre>String[] friends = {"Emily", "Bob" , "Cindy"};</pre>	An array of three strings
<pre>double[] data = new int[10];</pre>	ERROR: You cannot initialize a double[] variable with an array of type int[]

Table 1

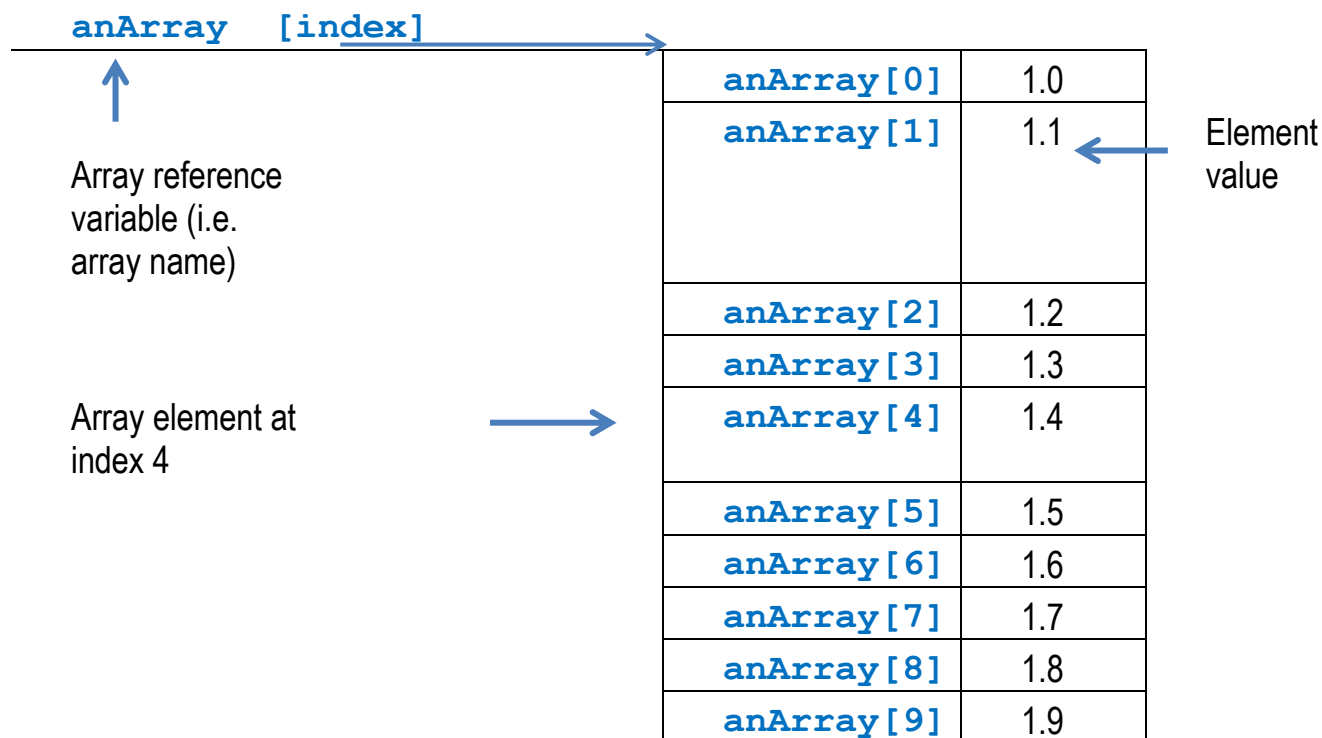
In order to create an array for the data type **double** with 10 elements and to allocate memory for the 10 doubles type the following code into the main method of your class:

```
//declares an array of the data type double and allocates memory for 10 doubles  
double [] anArray = new double[10];
```

This has declared the array. As yet no values have been assigned to the array. An array of a given size can have array members initialized on a one-to-one basis. If there are too few initializers for all members, the remaining members are initialized to 0.

Step 2: Initializing all 10 elements of the array

Each item in an array is called an *element*, and each element is accessed by its numerical *index*. As shown in the preceding illustration, numbering begins with 0. The table below outlines what we want our array to contain. Each individual element of the array will be initialized with a specific value, which is given below



Therefore, to initialize an element within an array, we would use the following syntax:

```
arrayRefVar[index] = value;
```

The square brackets are used to refer to each position within the array. Don't forget, because arrays start at 0, the first position in an array has the index number 0. To initialize our first element, type the following line of code underneath where you declared the array:

```
//initializing the first element of the array  
anArray[0] = 1.0;
```

This has now assigned the value 1 to position 0 in the array we have created. Continuing from the table above, we would initialize our second element with the following code

```
//initializing the second element of the array  
anArray[1] = 1.1;
```

You should now individually initialize the remaining elements of the array using the values specified in the table above. Remember to correctly reference the elements as you proceed. What happens if you try to initialize `anArray[10]` with a value?

Try it and see. As there are only 10 spaces in the array (indexed 0-9), trying to assign a value to the 11th space will cause an error.

You now have successfully declared and initialized your array. All elements should now have a value assigned to them, and can now be printed.

Step 3: Printing the array

The elements within the array can be printed using a print statement. The syntax for doing so would look similar to the code below:

```
Console.WriteLine("The element at index 0 in the array is "+anArray[0]);
```

Repeat the example above until you have a print statement for every element within the array. Remember to edit the text within the quotation marks so it is relevant to the element you are printing within that statement.

Save your work (CTRL+Shift+S) and then choose the Start  menu item (or press Ctrl+F5) .

Your output will be similar to that shown below:

```
The element at index 0 in the array is 1.0
The element at index 1 in the array is 1.1
The element at index 2 in the array is 1.2
The element at index 3 in the array is 1.3
The element at index 4 in the array is 1.4
The element at index 5 in the array is 1.5
The element at index 6 in the array is 1.6
The element at index 7 in the array is 1.7
The element at index 8 in the array is 1.8
The element at index 9 in the array is 1.9
```

TASK 2: ARRAYS AND LOOPS

In a real-world programming situation you would probably use one of the supported *looping constructs* to iterate through each element of the array rather than accessing the elements individually.

Step 1: Create an integer array

Create new project by selecting from the Visual Studio **File** menu, go to **New** and select **Project**. Name this project `Prac5Task2`.

Inside the Program class within the main method declare an array of the data type integer. Remember integers, written in your code as `int`, are whole numbers (for example, -35, 0, and 2048). For this task create an array for the data type `int` with 10 elements. As shown below:

```
//declares an array of the data type int and allocates memory for 10 ints
int [] intArray = new int[10];
```

Step 2: Write a for loop to populate the array

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. As an array is of a fixed length the `for` loop is ideally suited for processing. Look back to `Practical 4` where the syntax of for loops was introduced.

Type the following code beneath your array declaration to create a for loop:

```
for (int i = 0; i < 10; i ++){  
    intArray[i] = i;  
}
```

There are 10 elements which must be filled. The first element of the array is at `index 0` so the initializer variable, called `i` starts at 0. In order to continue the loop until we have reached ten, we will add one to `i` each time the loop is completed. The value that is being assigned to each position in the array is the current value of `i`.

Step 3: Write a for loop to print the array

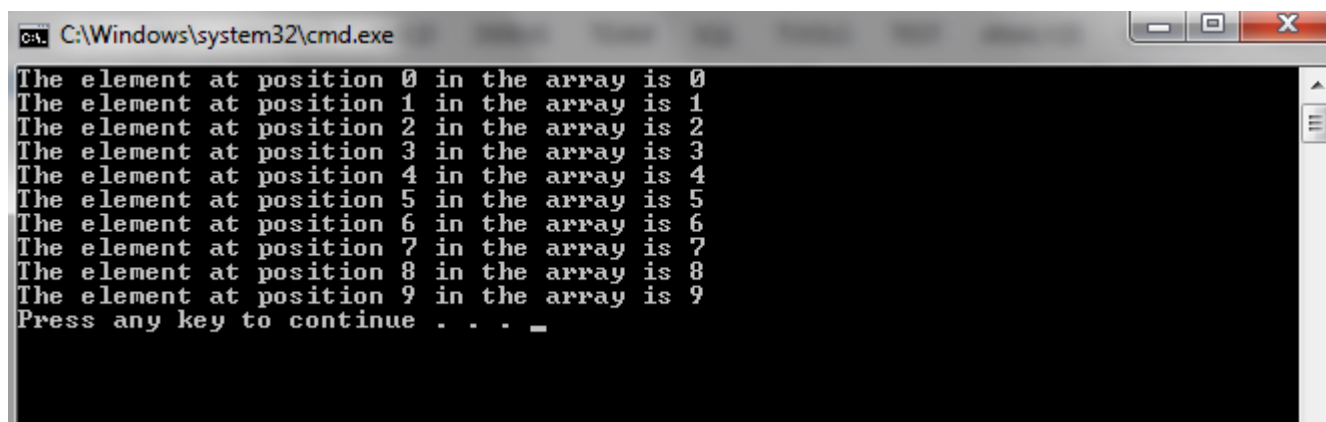
Next, we are going to create another for loop which will print out the array using a for loop.

```
// for loop to print the values in the array  
for (int i = 1; i < 2; 3){  
Console.WriteLine("The element at position " + i + " in the array is "  
+intArray[i]);  
}
```

Replace **1**, **2** and **3** in the for loop to correct code. Add some text to your print statement to explain what is happening as shown above.

Save the program, and run it. Think also about the scope of variables within the execution of this program particularly with reference to the variable `i`.

Expected output is shown below



```
C:\Windows\system32\cmd.exe  
The element at position 0 in the array is 0  
The element at position 1 in the array is 1  
The element at position 2 in the array is 2  
The element at position 3 in the array is 3  
The element at position 4 in the array is 4  
The element at position 5 in the array is 5  
The element at position 6 in the array is 6  
The element at position 7 in the array is 7  
The element at position 8 in the array is 8  
The element at position 9 in the array is 9  
Press any key to continue . . . _
```

TASK 3: SUM OF AN ARRAY

Often the elements of an array represent a series of values that may be used in a calculation for another purpose. For example, if the array represents exam grades, a professor may wish to total the elements of the array and use the sum later on. This task will outline how to find the sum of the elements of an array.

Step 1: Create a 10 element integer array

Create a new project from the Visual Studio **File** menu, go to **New** and select **Project**. Then enter a name for the new project such as `Prac5Task3`.

Use the table given below to create a 10 element integer array with the given values within your main method:

sumArray values

sumArray [0]	87
sumArray [1]	68
sumArray [2]	94
sumArray [3]	100
sumArray [4]	83
sumArray [5]	78
sumArray [6]	85
sumArray [7]	91
sumArray [8]	76
sumArray [9]	87

NOTE: When you know the values you wish to add to your array when you are creating, you can declare, create and initialize the array in one statement. The syntax is as follows:

```
typeName[] arrayRefVar = {value1, value2, value3,...value10};
```

This method of setting up an array uses curly brackets after the equals sign. In between the curly brackets, you type out the values that the array will hold. The first value will then be position 0, the second value position 1, and so on. Note that you still need the square brackets after the data type, but not the new keyword, or the repetition of the data type and square brackets. This is just for data types int, String, and char.

```
int[] sumArray = {87,68,94,100,83,78,85,91,76,87};
```

Step 2: Use a for loop to add the elements together

A 'for loop' will take each value from the array and add it to the cumulative total. Declare a variable total as follows:

```
int total = 0;
```

NOTE: As we are adding elements from an integer array, we should store the sum of the array in a variable of the data type integer as well.

Now we have somewhere to store the sum, we can create the 'for loop' which will add each element's value to the total. we know There are 10 elements which must be added to the total and the loop will iterate 10 times as in Task 2. Rather than hard wiring the value of 10 into the loop condition we can use the length.

Extra Information

When dealing with arrays, it is advantageous to know the number of possible elements that could be contained within the array, or the array's "**length**". This length can be obtained by using the array name followed by **.length**. If an array named numbers contains 10 values, the value of `numbers.length` will be 10.

**** You must remember** that the **length** of an array is the number of elements in the array, which is **one more than the largest subscript**.

Type this code into your program

```
for (int i = 0; i < sumArray.Length; i++)
```

Now you need to add a statement within your 'for loop' which will add each element to the total variable we declared earlier. This can be done as follows:

```
total += sumArray[i];
```

The += operator allows a cumulative total to be kept in the variable total. Can you think of another way to express the above statement without the += operator?

Step 3: Print the sum of the array

Finally print the total using a `Console.WriteLine()` statement.

EXERCISE 2

TASK 4: STORING USER INPUT IN AN ARRAY

Our next task uses arrays to store information that is collected via user input in a survey. This is a typical array-processing application, where we want to use `Console.ReadLine()` to accept input from the keyboard.

Step One:

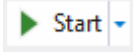
Create a new project from the Visual Studio **File** menu, go to **New** and select **Project**. Then enter a name for the new project such as `Prac5Task4`. Declare the array based on the information in the table below:

Array Information	
<code>arrayRefVar</code>	<code>studentNames</code>
<code>typeName</code>	<code>String</code>
<code>length</code>	<code>6</code>

Step 2: Allocate input to array elements

Write code to ask the user to enter six student names and allocate each name to the element in an array.

Step 3: Use a for loop to print all the names entered

Now we can use a 'for loop' to take all the data that has been entered and print it back out to the user. Create a 'for loop' which will iterate for the length of the array. Save your work (`CTRL +Shift+ S`) and choose the  menu item (or press `Ctrl+F5`). Check to ensure you get the output you expect based on what was input.

TASK 5: REVERSE PRINT ARRAY

An array can be printed in reverse order as by printing the value at the last index and decrementing to the first. This is useful for when you want to print data in reverse.

Step 1:

Create a new project from the Visual Studio **File** menu, go to **New** and select **Project**. Then enter a name for the new project such as `Prac5Task5`. Copy the code from `Task 4` and alter it so that the names will be accepted and then printed in reverse.

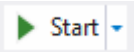
Step 2:

Consider the for loop for printing out the names:

```
for(int i = 1 ; i 2 ; i 3 )
    Console.WriteLine(studentNames[i]);
```

Replace **1,2** and **3** above by the correct syntax in order to print the array in reverse. (Hint: **1** is the length of the array; **2** is value of `i` will the loop stop; **3** is the operator for decrementing a value)

Step 3:

Save your work (`CTRL +Shift+ S`) and choose the  menu item (or press `Ctrl+F5`). Check to ensure you get the output you expect based on what was input.

TASK 7: PARTIALLY FILLED ARRAY

An array cannot change size at run time. This is a problem when you don't know in advance how many elements you need. In this situation, you must come up with a good guess on the maximum number of elements that you will need to store. In a typical program run, only part of the array will be occupied by actual elements. We call such an array a **partially filled array**.

Step 1:

From the Visual Studio **File** menu, go to **New** and select **Project**. This opens the New Project dialog box. Select a C# Console Application. Then enter a name for the new project such as `Prac5Task7`. Type the following code into the new class

```
class Program
{
    static void Main(string[] args)
    {
        const int LENGTH = 100;

        int[] valuesArray = new int[LENGTH];
        bool validInput = true;
        int counter = 0;

        while (validInput)
        {
            string input = Console.ReadLine();
            if (Int32.TryParse(input, out valuesArray[counter]))
            {
                counter++;
            }
            else
            {
                Console.WriteLine("Finished");
                validInput = false;
            }
        }

        for (int j = 0; j < counter; j++)
        {
            Console.WriteLine(valuesArray[j]);
        }
    }
}
```

We use a `Boolean` variable to check if we should continue the while loop thus continuing to accept input from the user.

The `if` statement that uses the `TryParse` method checks that the user has entered a number. If this is true then the loop continues. Otherwise we print a message informing the user the loop is closing and set the `Boolean` to `false` to exit the loop.

Step 2:

Save the class (CTRL + Shift+ S) and choose the **Start** menu item (or press Ctrl+F5). Check to ensure you get the expected outcome.

Notes on this task:

- You must keep a companion variable that counts how many elements are used (`counter`)
- At the end of the loop, `counter` contains the actual number of elements in the array
- To process the gathered elements, you again use the companion variable, not the array length, as it will cover the partially filled array rather than the entire length.

EXERCISE 3

This is the most challenging section of the practical and is designed to stretch and extend your knowledge and understanding of programming principals.

TASK 8: MULTI-DIMENSIONAL ARRAYS

The simplest form of the multidimensional array is the 2-dimensional array. A 2-dimensional array is, in essence, a list of one-dimensional arrays.

A 2-dimensional array can be thought of as a table, which will have x number of rows and y number of columns. Following is a 2-dimensional array, which contains 3 rows and 4 columns:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in array a is identified by an element name of the form a[i, j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is a two dimensional array with 3 rows and each row has 4 columns.

```
int[,] multiDim = new int[3, 4] { { 1, 2, 3, 4 }, { 1, 1, 1, 1 }, { 2, 2, 2, 2 } };
```

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example:

```
int val = multiDim[2,3];
```

This statement will set the variable val to the value located at the index provided.

Now we will print out the contents of the array.

```
Console.WriteLine(multiDim[2,3]);
```

This statement will print the value located at the index specified but it can often be useful to print the entire contents of the array.

Create your multidimensional array as shown above and provide as many columns and rows as you like as well as supplying values for the array.

Copy the code shown below to achieve the desired results.

We use a nested for loop to iterate through the index of the array.

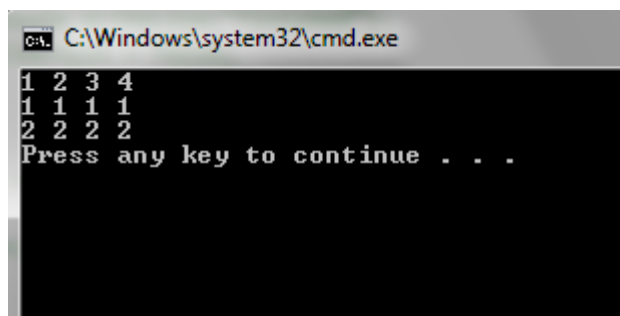
The outer loop is used to loop through the rows and the inner loop is used to loop through the columns.

```
int[,] multiDim = new int[3, 4] { { 1, 2, 3, 4 }, { 1, 1, 1, 1 }, { 2, 2, 2, 2 } };

for (int p = 0; p < multiDim.GetLength(0); p++)
{
    for (int k = 0; k < multiDim.GetLength(1); k++)
    {
        Console.Write(multiDim[p,k]+" ");
    }
    Console.WriteLine();
}
```

The `GetLength()` method is used to get the length of a specific dimension of the array. `GetLength(0)` will relate to the number of rows and `GetLength(1)` will return the number of columns in this example.

The expected output is shown below.



```
C:\Windows\system32\cmd.exe
1 2 3 4
1 1 1 1
2 2 2 2
Press any key to continue . . .
```

For this exercise, you will be required to create a program that reads a sequence of values and adds them to an array. Your program will firstly find the largest value in the array, print out the values in the array indicating the largest value. Finally your program will find the smallest value in the array, print out the values in the array indicating the largest value

TASK 9: LARGEST AND SMALLEST VALUE IN THE ARRAY

Step 1: Create the array and the length/currentSize variables

- Create the final `LENGTH` variable
- Create the array: typeName `double`, size set to variable `LENGTH`
- Create the companion variable `currentSize`
- Create two double variables; `currentLargest` and `currentSmallest`.

Step 2: Read the input from scanner using a while loop then print array

- Create a while loop to take input `while currentSize < array LENGTH`
 - Within loop, assign `input` to current element of the array
 - Within loop, increment `currentSize` variable by one

Step 3: Find the largest value

- Assign to the variable `currentLargest` the value stored in the first array position
- Use a for loop to iterate through the array and compare values
- Use an if statement to compare the current element and the `currentLargest`
- If the current element is greater than `currentLargest`, assign its value to `currentLargest`

Step 4: Print all values, marking the largest, using a for loop

- Loop to print the array and use an if statement to find the largest value i.e. the value equal to `currentLargest`
- Use the following to mark largest value
 - `Console.WriteLine(" <== largest value in Array");`

Step 5: Find the smallest value

- Assign to the variable `currentSmallest` the value stored in the first array position
- Use a for loop to iterate through the array and compare values
- Use an if statement to compare the current element against `currentSmallest`
- If the current element is less than `currentSmallest`, assign its value to `currentSmallest`

Step 6: Print all values, marking the smallest, using a for loop

- Loop to print the array and use an if statement to find the largest value i.e. the value equal to `currentSmallest`
- Use the following to mark smallest value
 - `Console.WriteLine (" <== smallest value in Array");`

Step 7: Run

Save the class (CTRL + S) and choose the Start menu item (or press Ctrl+F5) . Ensure your results are, as you would expect depending on your input.

TASK 10: TIMES TABLES USING MULTIDIMENSIONAL ARRAYS

This aim of this task is to produce columns of the times tables similar to that shown below.

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

The first column is the 1 times table, the second is the 2 times table and so on. The program will ask for user input to determine a value for the number of rows and the number of columns in the table. The value of rows will determine to what level the particular times table is generated. The value of columns will determine how many times tables are generated.

First declare your variables which will be used to store the rows and columns. Use `Console.ReadLine()` to get input from the user. Then declare your two dimensional array and set the dimensions to your rows and columns variables as shown below.

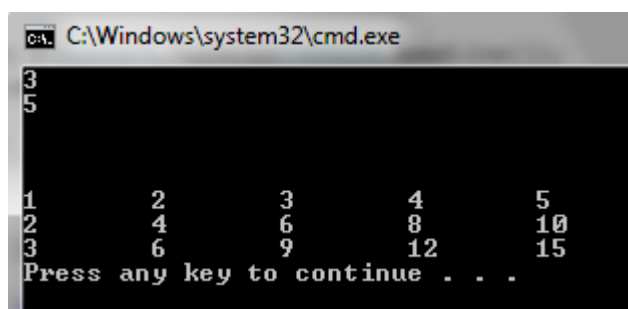
```
int[,] multiDim = new int[rows, columns];
```

Use a nested for loop to iterate through your array and perform the calculation shown below to fill the array with the appropriate values. Use the `GetLength()` method shown in the previous exercise to specify how many times you will iterate through each loop.

```
for (int p = 0; p < multiDim.GetLength(0); p++)
{
    for (int k = 0; k < multiDim.GetLength(1); k++)
    {
        multiDim[p, k] = (p + 1) * (k + 1);
    }
    Console.WriteLine();
}
```

Now use another nested for loop to print the contents of your array in the appropriate format. This will be similar as the previous task.

The expected output is shown below.



```
C:\Windows\system32\cmd.exe
3
5
1      2      3      4      5
2      4      6      8      10
3      6      9      12     15
Press any key to continue . . .
```

Now you will use a `foreach` loop which will iterate through the array in the same way as a nested for loop. The results will be the same but the method is slightly different. The syntax for using a `foreach` loop is shown below. A `foreach` repeats a group of statements for each element in an array in the same way that a normal for loop does.

```
foreach (int i in multiDim)
{

}
}
```

For each integer in our multidimensional array we will perform some action, whatever we write inside the curly brackets. Input the code shown below between the brackets.

```
Console.Write(i+"\t");

if (i == multiDim.GetLength(1))
{
    Console.WriteLine();
}
```

First the element is printed out followed by a tab. Included is a simple if statement which uses the `GetLength` method to check if we have reached the end of a row. If this is true then we want to take a new line. This simply prints the array in a matrix like layout. The output should be similar to the example shown above.

EXERCISE 4

TASK 11: CREATING AN ARRAYLIST

We will develop a program in which we will create an `ArrayList` of the Object type `String` and fill it with an undetermined amount of `String` objects.

If the number of elements are known, or small fixed size, or where efficiency in using data primitives is important, arrays are better as using an `ArrayList` will cause your program to run slower.

An `ArrayList` can only be used to hold Object types and not data primitive types (e.g. `double`). To use a data primitive type in an `ArrayList` put it inside an `Object` or use one of the wrapper classes (e.g. `Double`).

Step 1:

Open the Visual Studio **File** menu, and then select **New**. Then click on **Project** and name it e.g. `Prac05Task10`.

Declaring an `ArrayList` is basically the same as declaring any other type of `Object` or primitive with the difference being that you must state, within the `<>` brackets what type the `ArrayList` is. Therefore to declare an `ArrayList` of type `String` you would type the following:

```
ArrayList myList = new ArrayList();
```

We are going to use the `Random` class to generate a random integer between 1 and 100 to determine the amount of elements that we are going to store. Use the following code to achieve this outcome:

```
Random random = new Random();  
  
int randomNumber = random.Next(0, 100);
```

Step 2: Adding data to the ArrayList

Like an array the contents of an `ArrayList` are known as elements. However unlike an array we do not need to know the element number of an element that is being added to an `ArrayList` as data is added to the end of the list in a contiguous manner. If we want to add the element to a specific place this can also be done and this will be explained later.

To add data to an `ArrayList` we can use two different methods. The `Add` method adds our object to the end of the array list. The `Insert` method can be used to add an element to our array list at a specific index so we supply this index as well as the object we want to add.

```
myList.Add("objectToAdd");  
  
myList.Insert(2, "objectToAdd");
```

Use a for loop to add elements to the `ArrayList` until the counter has reached the value of the previously declared `int` `randomNumber`.

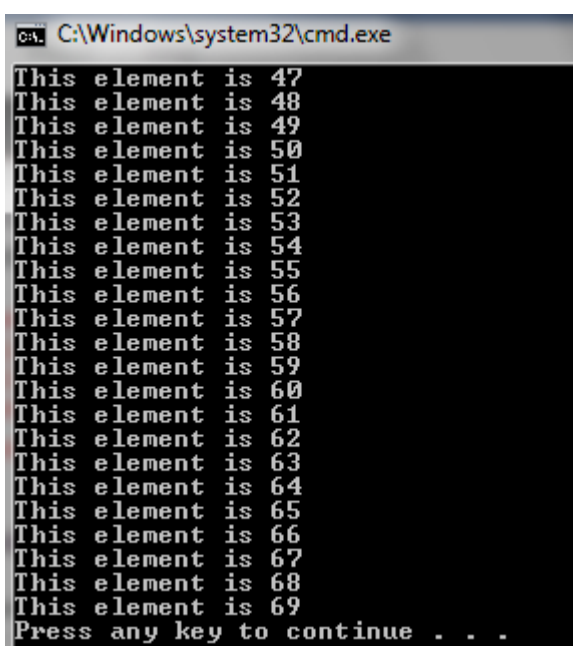
```
for (int i = 0; i < randomNumber; i++)  
{  
    myList.Add("This element is "+i);  
}
```

You now have successfully declared and initialized your `ArrayList`. All elements should now have a value assigned to them, and can now be used for printing. Your `ArrayList` size should have the same value as the previously declared `int randomNumber`.

Step 3: Printing the contents of the `ArrayList`

Finally, we can reference the elements within the `ArrayList` within a print statement to print out the values they contain. The syntax for doing so would look similar this:

```
for (int i = 0; i < myList.Count; i++)
{
    Console.WriteLine(myList[i]);
}
```



```
C:\Windows\system32\cmd.exe
This element is 47
This element is 48
This element is 49
This element is 50
This element is 51
This element is 52
This element is 53
This element is 54
This element is 55
This element is 56
This element is 57
This element is 58
This element is 59
This element is 60
This element is 61
This element is 62
This element is 63
This element is 64
This element is 65
This element is 66
This element is 67
This element is 68
This element is 69
Press any key to continue . . .
```

TASK 12: ITERATORS

For the following couple of tasks we are going to explore the different ways of looping through `ArrayLists` and the different kinds of results that we can achieve. An `ArrayList` will be randomly filled with the values `Triangle`, `Rectangle`, `Circle` or `Square`. The `ArrayList` will then be printed out and then the first and last occurrence of a particular value will be found. Finally an element will be removed from the `ArrayList`.

Step 1: Create a String array

Create a **New Project** from the Visual Studio **File Menu > New> Project**. This opens the **New Project** dialog box. Then enter the name `Prac5Task11` as the name of your Project. Declare a `String` array using the information in the variable declaration table below:

Variable Information

Name	shapes
dataType	String[]
Size	4
Values	Triangle, Rectangle, Circle, Square

```
String [] shapes = new String[]{"Circle", "Triangle", "Rectangle", "Square"};
```

Step 3: Declare an ArrayList of type String

Look back to the previous task if you are stuck on the syntax for this statement, call the `arraylist` `shapeList`. After declaring your `arraylist` you will need to generate a random number between 1 and 100. This was covered in the last task so look back if you need to refresh your memory. Declare an `int` variable and call it `randomShapeSelection`. This will be used to hold the index of the shape that is randomly chosen from the shapes array.

Step 4: Populate the ArrayList

Begin a `for` loop with the loop running as long as your iterator is less than the `randomNumber` between 1 and 100. Inside the loop you will generate another random number between 1 and 4 which will relate to one of the shapes stored in the shapes array.

```
for (int i = 0; i < randomNumber; i++)
{
    randomShape = random.Next(4);
}
```

Then add the randomly selected shape to your `arraylist` using this line

```
shapesList.Add(shapes[randomShape]);
```

Step 5: Print the contents of the ArrayList

Print the `arraylist` to check that the items we successfully added. You will use a `foreach` loop to print each element of the array.

The syntax for a `foreach` loop is recapped below.

```
foreach(var i in shapesList)
{
    Console.WriteLine(i);
}
```

You should receive somewhat similar output as is shown below but obviously the number of elements will be random.


```
C:\Windows\system32\cmd.exe
```

```
Square  
Square  
Rectangle  
Circle  
Circle  
Triangle  
Circle  
Triangle  
Circle  
Triangle  
Square  
Triangle  
Rectangle  
Circle  
Rectangle  
Triangle  
Square  
Circle  
Triangle
```