Practical Task 3.1

(Pass Task)

Submission deadline: 10:00am Monday, April 13
Discussion deadline: 10:00am Saturday, May 2

General Instructions

This practical task introduces you to ***arrays*** and ***lists*** that allow you to group, store and manage related data. Arrays are fixed in size and are therefore more efficient than a ***List***. A `List` is a more flexible data structure that allows you to manage undefined number of records.

1. In this part of the task, you will create an array with a data type of double. The array will have 10 elements and you will individually initialise each one. After you have initialised the elements, you will print each element to the console with reference to the array.

   The way to declare an array is very similar to how we would do this for a variable. We must give it a type using the following format:

   ```
   typeName[] arrayRefVar;
   ```

   Here, `typeName` can refer to any of the primitive data types or a class name, which can be user defined. To construct the array, we must apply the '***new***' operator as follows:

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

   The above statement does two things:

   - It creates a new array of type `typeName` and of length `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 follows:

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

   STEP 1: Start a new C# Console Application project. You may keep the name of the main class as `Program`, which is the default name. Create an array of data type double with 10 elements by typing the following code into the main method of your `Program` class:

   ```
   //declares an array of type double with 10 elements
   double[] myArray = new double[10];
   ```

   This will declare the array and set the value of each element to zero (as a default value). You can now start assigning your own values to the each element, but remember that you cannot have any more than 10 elements in this array. Each item in an array is called an ***element*** and each element is accessed by its ***numerical index***. Also note that numbering begins at 0. Therefore, to initialize an element within an array, we must use the following syntax:

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

   The square brackets are used to refer to the position of each element within the array. Do not forget that arrays start at 0, the first position in an array has the index number 0. To assign a value to the first element, type the following line of code underneath where you declared the array:

   ```
   //assigning the first element of the array
   myArray[0] = 1.0;
   ```

   The first element of the array, at position 0, has now been assigned the value 1.0. You now can proceed with the second element by typing the next line of code:

   ```
   //assigning the second element of the array
   myArray[1] = 1.1;
   ```

The table provided below outlines what we want your array to contain. You should now individually assign the remaining elements of the array using the values specified in the table. Remember to correctly reference the elements as you proceed.

index	0	1	2	3	4	5	6	7	8	9
value	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9

STEP 2: You will need now to print all elements of the array to the console. Use the following code adding it underneath the last assignment to position 9 in the array:

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

Repeat the code above for each element in the array. Remember to edit the text within the quotation marks so it is relevant to the element you are printing within that statement.

Compile, run and check the program for potential errors. What happens if you try to assign a value to index 10? 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. This is to be a main error you will observe when working with arrays.

2. As you should note, there is a lot of repeated code and it would be much easier to use a loop to iterate through each element of the array rather than accessing them individually. This task steps you through using loops to assign values and print them to console.

   STEP 1: Continue with the current project and, within the main method of the Program class, declare an array of data type integer that will hold 10 elements.

```
//declares an array of type integer with 10 elements
int[] myArray = new int[10];
```

   Write a ***for loop*** to populate the array by typing the following code beneath your array declaration:

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

   There are 10 elements in the array, which must be assigned a value. The first element of the array is at index 0 so the initializer variable of the ***for loop***, designated as **`i`**, starts at 0. In order to fill all elements of the array, we want the loop to continue until it reaches 9 (i.e. less than 10). We will add 1 to **`i`** each time the loop is executed. The value that is being assigned to each element of the array is the current value of **`i`**. This loop should produce the following content of the array.

index	0	1	2	3	4	5	6	7	8	9
value	0	1	2	3	4	5	6	7	8	9

   STEP 2: You must now create another ***for loop*** in order to print out the array. Type the following code beneath the loop you wrote to assign values.

```
// for loop to print the values in the array
for (int i = 0; i < 10; i++)
{
    Console.WriteLine("The element at position " + i + " in the array is " + myArray[i]);
}
```

   Compile, run and check the program for potential errors.

3. 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 for a student, a tutor may wish to total the elements of the array to calculate the average mark attained by that student. This task will step you through coding this.

Continue with the current project and use the information from the table below to create a 10 element integer array.

index	0	1	2	3	4	5	6	7	8	9
value	87	68	94	100	83	78	85	91	76	87

When you know the array values you need at the point of declaration, then you can declare, create and assign values in one statement. The syntax is as follows:

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

Type in the following code to declare, create and assign the values.

```
int[] studentArray = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
int total = 0;

for (int i = 0; i < studentArray.Length; i++)
{
    total += studentArray[i];
}
```

To add all the values together we will need a variable to hold the running total, e.g. `total`. This needs to be declared outside of the ***for loop*** and initialised to zero. If declared inside the loop, then it is only accessible inside the loop, but we will need the information stored in the variable outside the loop. Within the ***for loop*** we use `studentArray.Length` which is an inbuilt method of an array that provides the length of it. It is better programming practice to use `array.Length` as we do not need to worry about knowing how many values are actually stored in the array. Finally, we use '**+=**' operator to say that

```
total = total + studentArray[i]
```

Now let us print out some values of the array, namely the total, the number of elements in the array, and the average mark. To do this, type the following code after the ***for loop***.

```
Console.WriteLine("The total marks for the student is: " + total);
Console.WriteLine("This consist of " + studentArray.Length + " marks");
Console.WriteLine("Therefore the average mark is " + (total/studentArray.Length));
```

Compile, run and check the program for potential errors.

4. Your next task is to utilize 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 and store it in an array.

   Continue with the current Console Application project and declare the array based on the information in the following table.

variable name	studentNames
data type	String
length	6

   Write code asking the user to enter six student names and assign each name to the next element in the declared array. Then use a **for loop** to print all the entered names. Compile, run and check the program for potential errors.

5. Your next exercise is to write a program to read a sequence of values and add them to an array. You will then find the largest and smallest values of the array, print these to screen and print the contents of the array.

   Continue with the current project and carry out the following instructions to set up your program.

   - Create a one-dimensional array of type double and length 10.
   - Create a `currentSize` variable of type `int` and initialise it to zero.
   - Create two variables of type double named `currentLargest` and `currentSmallest`.

   Using these variables, write code to read input from the console and print the array. Then add program code to find the largest value by implementing the following instructions:

- Assign to the variable `currentLargest` the value stored in the first position in the array.
- Use a ***for loop*** to iterate through the array. In the for loop, use an ***if statement*** to compare the current element of the array to the value held in the `currentLargest` variable. If the current element is greater than the `currentLargest`, assign its value to the `currentLargest`. In the ***for loop***, print out each element of the array.
- Print the largest value of the array to the console.

Similarly, find the smallest value of the array. Compile, run and check the program for potential errors.

6. In this part, we will look at multi-dimensional arrays and also start to look at other functions that we can perform on an array. The simplest form of the multidimensional array is the 2-dimensional array. A 2-dimensional array is, in essence, an array of arrays. The easiest way to consider a 2-dimensional array is as a table. The table will have x number of rows and y number of columns. The following figure illustrates a 2-dimensional array, which contains 3 rows and 4 columns. It also shows how to generally access each of the values (or cells).

	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 **myArray** is identified by an element name of the form

```
myArray[row,column]
```

Multidimensional arrays may be initialised by specifying bracketed values for each row. For example,

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

is an equivalent of the table below.

	Column 1	Column 2	Column 3	Column 4
Row 0	1	2	3	4
Row 1	1	1	1	1
Row 2	2	2	2	2

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

```
int value = myArray[2,3];
```

This statement will set the variable value to the value located on the second row in the third column.

Continue with the current project and declare the array based on the information from the table above. To step through the multidimensional array, you will need a nested ***for loop***. The first loop will step through each row, and the second loop will step through each column. Type the following code to print out each element of your array

```
for (int i = 0; i < myArray.GetLength(0); i++)
{
    for (int j = 0; j < myArray.GetLength(1); j++)
    {
        Console.Write(myArray[i, j] + "\t");
    }
    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.

Compile, run and check the program for potential errors.

The .NET Framework has another structure that handles many of the issues associate with arrays a little easier. This is a ***List***, which is commonly referred to as a ***collection***. It is considered to be an object and

once created provides us with the ability to add an item, remove an item (from any location in the List), peek at an item and also move an item from one place in the list to another. These functions are not easily applied to an array, and therefore a `List` provides more flexibility depending on the data you want to "collect". However, if the number of elements in the `List` is known, or if the `List` is of a small fixed size, or if the efficiency in using primitive data types is important, it is better to use arrays. In this case, using a `List` will cause your program to run slower.

Your task is now to create a `List` to hold an undetermined number of names. Consider setting up a new list of students at the start of term, you will need all names but until they arrive you are unsure how many students there will be. A `List` can be used to hold ***object references*** and ***primitive data types*** (e.g. double).

Continue with the current project and declare a `List`. This process is very similar to declaring any other object. We must tell it what object (give it a class name) you are creating, i.e. a `List`, and provide it with a type, e.g. String, and we instantiate an instance of the object using the keyword '***new***'. Declare a `List` of type String by typing the following code:

```
List<String> myStudentList = new List<String>();
```

In this exercise, we are going to use the `Random` class to generate a random integer between 1 and 12 to determine the amount of elements we are going to store. To set up the random value, for the number of students in the class, type the following code:

```
Random randomValue = new Random();
int randomNumber = randomValue.Next(1, 12);
```

Like an array, the contents of a `List` are known as elements. However, unlike an array, we do not need to know the position or index that an element is being added to. For a `List`, data is added to the end of it. If we want to add data at a particular position in the `List`, then we can also achieve this. Therefore, we have two different methods we can use when adding information to a `List`.

The two different methods for adding data to a `List` are as follows:

- The `Add` method: this adds data to the end of the `List`, using `myList.Add("objectToAdd")`.
- The `Insert` method: this adds data to the `List` at a specific index. Therefore, we supply the `Insert` method with the index as well as the data we want to add, using `myList.Insert(2,"objectToAdd")`.

In this example, we will use the `Add` method to add data to the end of the list. We will use a ***for loop*** to add elements to the `List` until the counter has reached the value of the previously declared variable `randomNumber`. Implement this via the following code:

```
Console.WriteLine("You now need to add " + randomNumber + " students to your class list");
for (int i = 0; i < randomNumber; i++)
{
    Console.Write("Please enter the name of Student " + (i + 1) + ": ");
    myStudentList.Add(Console.ReadLine());
    Console.WriteLine();
}
```

You now have successfully declared and initialised your `List`. All elements have also been assigned a value. Your `List` size should have the same value as the previously declared variable `randomNumber`, but you can check this using the `Count` property. Therefore, to obtain how many elements are in an array we using `Length`, and for a `List` we use `Count`.

Use a ***for loop*** to print out all of the data held in your `List`. Compile, run and check the program for potential errors.

7. In the `Program` class, add a function `FuncOne` that accepts as input an arbitrary array of data type integer. The function must check the size of the array, and if the number of elements is greater than 10, it needs to return the number of even elements, otherwise the total product of odd elements. Add an appropriate program code in the main method of the class to check the function for potential errors.

8. In the `Program` class, add a function `FuncTwo` that accepts as input a List of data type double. The function must compute the average value across all elements of the list and transform the given list by subtracting the average value from every element of the collection. Add an appropriate program code in the main method of the class to check the function for potential errors.
9. In the `Program` class, add a function `FuncTree` that accepts as input an arbitrary 2-dimensional array of data type integer. The function must covert the input into a one-dimensional array by copying all the values that are multiple 3. It must consider elements column by column, in ascending order, and for every column examine elements in the ascending order of the row index. It must return the resulting one-dimensional array. Add an appropriate program code in the main method of the class to check the function for potential errors.
10. In the `Program` class, add a function `FuncFour` that accepts as input a one-dimensional array of data type integer. The function must iterate through the elements of the array and produce a multiplication table based on the values recorded in the array. The resulting multiplication table returned by the function must be in the form of a two-dimensional array. Add an appropriate program code in the main method of the class to check the function for potential errors.

Further Notes

- Study the concept of a data collection and how to use it in your programs by reading Sections 3.2 of SIT232 Workbook available in CloudDeakin → Resources.
- The following links will give you more insights about arrays, lists, and the way to use them:
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections
  - https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1
- In this unit, we will use Microsoft Visual Studio 2017 to develop C# programs. Find the instructions to install the community version of Microsoft Visual Studio 2017 available on the SIT232 unit web-page in CloudDeakin at Resources → Software → Visual Studio Community 2017. You however are free to use another IDE, e.g. Visual Studio Code, if you prefer that.

Marking Process and Discussion

To get your task completed, you must finish the following steps strictly on time.

- Make sure that your programs implement the required functionality. They must compile and have no runtime errors. Programs causing compilation or runtime errors will not be accepted as a solution. You need to test your program thoroughly before submission. Think about potential errors where your program might fail.
- Submit the expected code files as an answer to the task via OnTrack submission system. Cloud students must record a short video explaining their work and solution to the task. Upload the video to one of accessible resources, and refer to it for the purpose of marking. You must provide a working link to the video to your marking tutor in OnTrack.
- On-campus students must meet with their marking tutor to demonstrate and discuss their solution in one of the dedicated practical sessions. Be on time with respect to the specified discussion deadline.
- Answer all additional questions that your tutor may ask you. Questions are likely to cover lecture notes, so attending (or watching) lectures should help you with this **compulsory** interview part. Please, come prepared so that the class time is used efficiently and fairly for all students in it. You should start your interview as soon as possible as if your answers are wrong, you may have to pass another interview, still before the deadline. Use available attempts properly.

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Remember that this is your responsibility to keep track of your progress in the unit that includes checking which tasks have been marked as completed in the OnTrack system by your marking tutor, and which are still to be finalised. When marking you at the end of the unit, we will solely rely on the records of the OnTrack system and feedback provided by your tutor about your overall progress and quality of your solutions.