Programming in C# Lab courses

Overview

This laboratory exercises are meant for people who want to start developing programmer skills but has very little or no professional experience programming in C#.

In this lab exercises we´ll learn the basics in C# programming. We´ll cover topics like how to retrieve user´s info, perform basic arithmetic operations, how to use arrays, and how to call external services (API) among other fundamental topics that will be discussed.

Practice makes perfect, so they say. Well, let´s prove it! 😉

Requirements

- For this lab exercises you will need some or very little experience with programming.
- You´ll need to install Visual Studio 2019 (IDE) https://visualstudio.microsoft.com/

General Objectives

In this lab sessions we will put in practice the concepts that we learned in class, with this lab sessions students will also become familiar with the fundamental concepts and syntax of C#. At the end of each module, you will be able to find a series of tasks that will challenge your knowledge so you can develop solutions in C# on your own.

Module 01: Receive Data and perform arithmetic operations.

Lab Setup

Estimated time: **50-60 min.**

Introduction

C# provides several operators. Typically, you can overload those operators (specify the operator behavior for the operands of a user-defined type).

The simplest C# expressions are literals (for example, integer and real numbers) and names of variables.

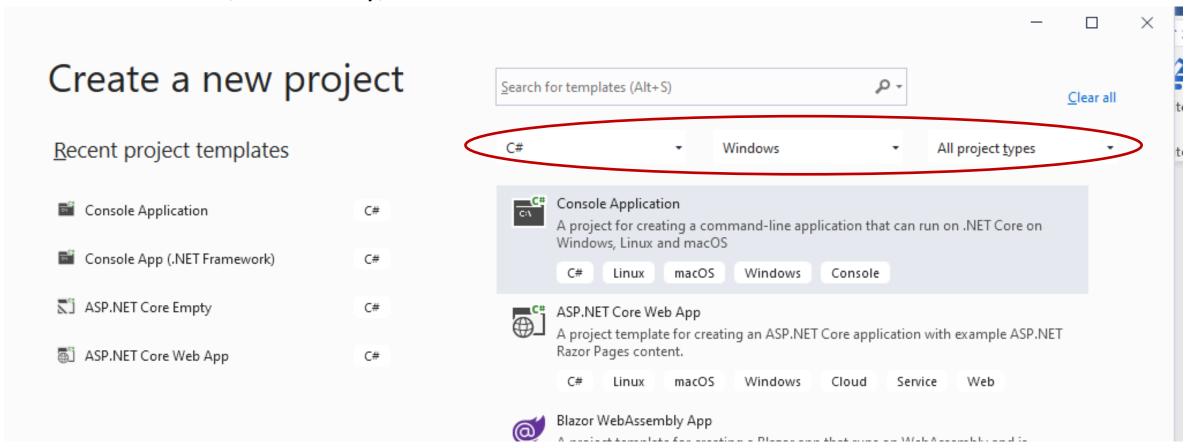
Objectives

Welcome to the first of many exercises that will test the knowledge acquired during your Introduction to C# classes. In this very first lab you will:

a. Learn how to create a new project in Visual Studio.
b. Put in practice the basics of the C# syntax.
c. Get familiar with your workspace in Visual Studio.
d. Do basic arithmetic operations.
e. Write to the console of your app.

Exercise 01: Use `Console.WriteLine` to display information into your console.

1. Open Visual Studio 2019.
2. Select Create a New Project.
3. From the top right drop-down menu, look for **C#**, in the next drop-down menu choose **Windows**, and finally, choose **Console**.



Create a new project

Search for templates (Alt+S)

Clear all

Recent project templates

C# | Windows | All project types

Console Application C#
Console App (.NET Framework) C#
ASP.NET Core Empty C#
ASP.NET Core Web App C#

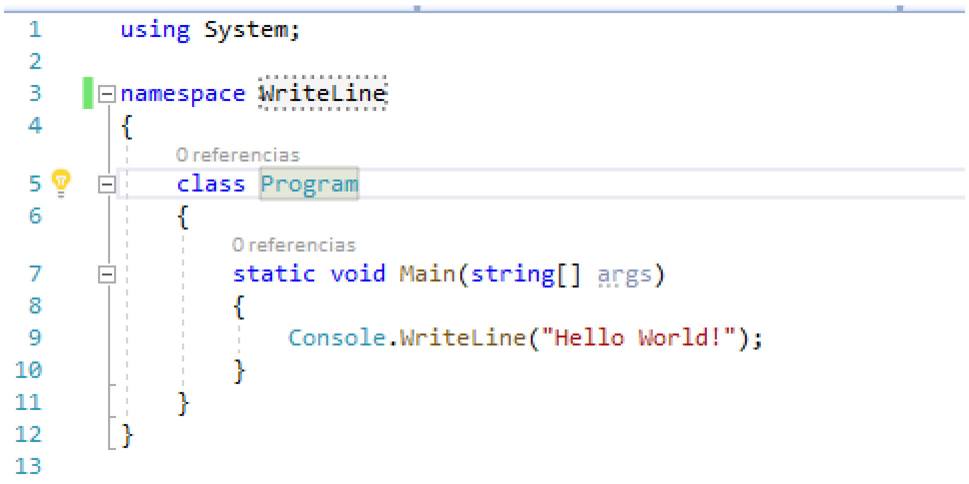
Console Application
A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS
C# Linux macOS Windows Console

ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

Blazor WebAssembly App

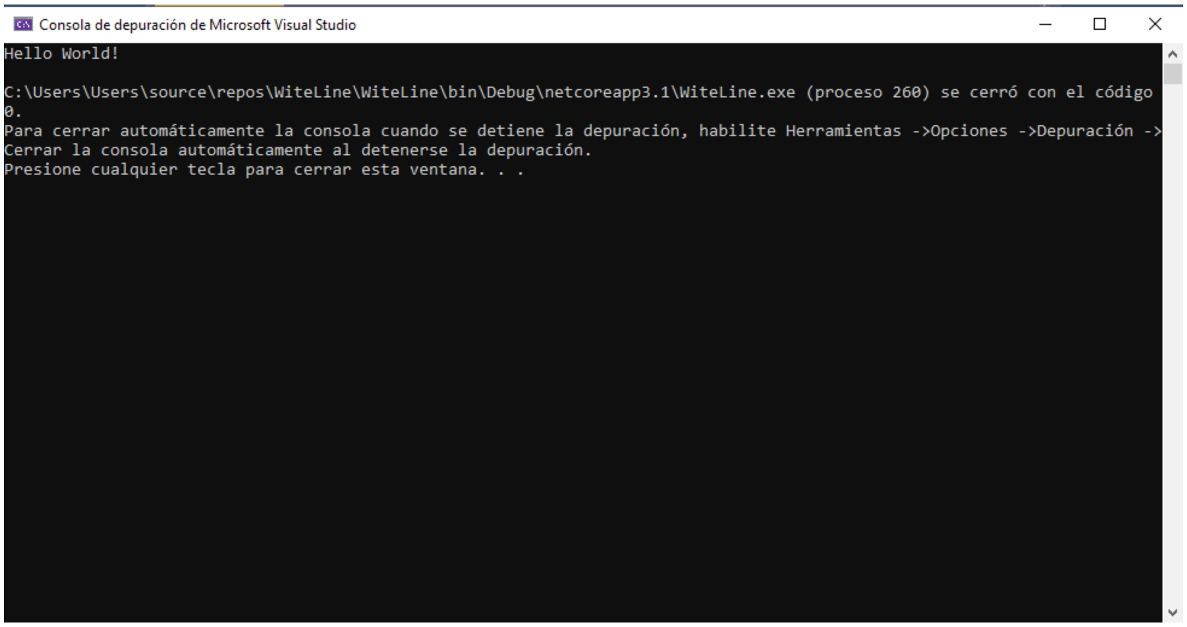
4. Now, choose the first filtered option: **Console Application** .NET core and click next.
5. In the next window, you can **change the Project´s name to something you like** and click **next**.

6. Choose **.NET core 3.1 (long term compatibility)** and click create.
7. Your Workspace should now look something like this:



```
using System;

namespace WriteLine
{
    0 referencias
    class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

8. To compile this code, please click on the **play button** located in the top of your Visual Studio window. You´ll now see how the console (will run and open up a black window showing the Hello World! Message:



Consola de depuración de Microsoft Visual Studio

```
Hello World!

C:\Users\Users\source\repos\WiteLine\WiteLine\bin\Debug\netcoreapp3.1\WiteLine.exe (proceso 260) se cerró con el código
0.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

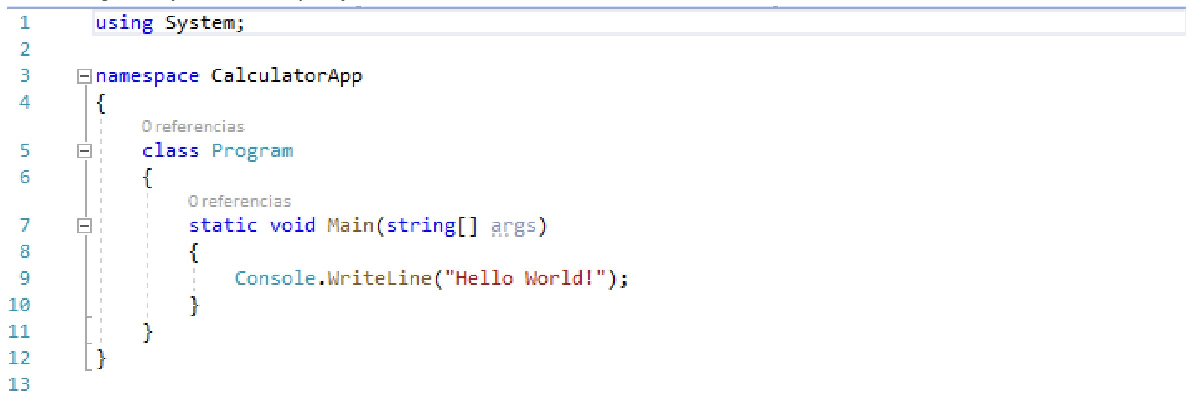
9. Close the console window, and back in Visual Studio please select all the code and replace it for the following code:

```
using System;

    public class Exercise1
    {
        public static void Main()
        {
            System.Console.WriteLine("Hello World");
        }
        }
```

10. Once you have changed the code, please click once again in the play button (or you can also hit F5 key to start compiling and running your app).
11. As you can see, both codes do the exact same task: to display a piece of text in the console. The only difference between this two, is that the code shown in step 9 is simplified.

Exercise 2: Use `Console.ReadLine` to read inputs from users.

1. Open Visual Studio 2019.
2. Create a new project and change the name of the project to something you like, i.e., CalculatorApp (please follow steps 2-6 from the previous exercise).
3. Once again, your new project should look like this:



```
using System;

namespace CalculatorApp
{
    0 referencias
    class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}

```

4. **Delete all code** created in this new file and please paste the following one:

```
using System;

public class Exercise02
{
    public static void Main()
    {
        int firstNum, secNum, sum;

        Console.Write("Introduce first number: ");
        firstNum = Convert.ToInt32(Console.ReadLine());
        Console.Write("Introduce second number: ");
        secNum = Convert.ToInt32(Console.ReadLine());
        sum = firstNum + secNum;

        Console.WriteLine("The sum of {0} and {1} es {2}",
            firstNum, secNum, sum);
    }
}
```

5. In this example, we´re creating a simple calculator to help you understand how input is retrieved from a user by using the `Console.ReadLine` command and how you can perform simple arithmetic operations using C# with these inputs.
6. The `int` command allow us to create integers inside our calculator.
7. The `Convert.ToInt32` command will convert the input value into numerical inputs.
8. You can now save your project and close Visual Studio for the next lab.

Challenge

Create a simple calculator that allows you to sum even numbers only. If an odd number is used, display an error message in the console.

Module 02: Arrays.

Lab Setup

Estimated time: **50-60 min.**

Introduction

In class, you learned that arrays in C# are meant to store multiple variables of the same type. First you declare an array by specifying the type of it´s elements. It´s important to remark that Arrays can be single-dimensional, multidimensional, aor jagged and that an array with n elements is indexed from 0 to n-1.

Objectives

Arrays are a very useful way to manage your information, in this lab exercise you will learn:

a. Create Arrays.
b. Sort the information of the array and then display it in the console
c. Change the array sort.

Exercise 01: Display and sort Arrays.

We´ll use the 3 different methods of sorting information inside an array.

1. Open Visual Studio 2019.
2. Create a **new project** choosing the **same settings we used in the 1st module** and change the name of the project `ArrayExample`.
3. Please delete all the preloaded text in the project and we´ll begin with a blank page.

4. You can copy and paste the following code which is where we´ll begin writing our first Array exercises.

```
using System;

namespace ArrayExample

{

        class Program

        {

                static void Main(string[] args)

                {

                

                }

        }

}
```

5. From now on, all text will be written inside the brackets of our `static void Main(string[] args).` The zone for writing the new code is marked with a yellow rectangle in the text above.

6. We´ll begin this exercise by declaring our first Array, which is made of 5 numbers. In step we´ll also ask the app to display the 5 numbers contained in our Array.

```
int[] array = new int[5] { 1, 4, 2, 3, 5 };
Console.WriteLine("---Default Array Elements---");
foreach (int i in array)
{
        Console.WriteLine(i);
}
```

7. Now we´ll use the sort method to display our elements in numerical order, and then display them in the console. You can copy and paste the following text right below the code we just wrote in step 6.

```
Array.Sort(array);
Console.WriteLine("---Elements After Sort---");
foreach (int i in array)
{
        Console.WriteLine(i);
        }
```

8. The reverse method will allow us to re-arrange elements inside an array. In this step, we´ll also ask our app to display the “reversed” arrangement of numbers in the console.

```
        Array.Reverse(array);
        Console.WriteLine("---Elements After Sort---");
        foreach (int i in array)
        {
                Console.WriteLine(i);
        }
```

9. Finish the task and close the app.

```
        Console.WriteLine("Press Enter Key to Exit..");
        Console.ReadLine();
```

10. In the end, your code should look like this:

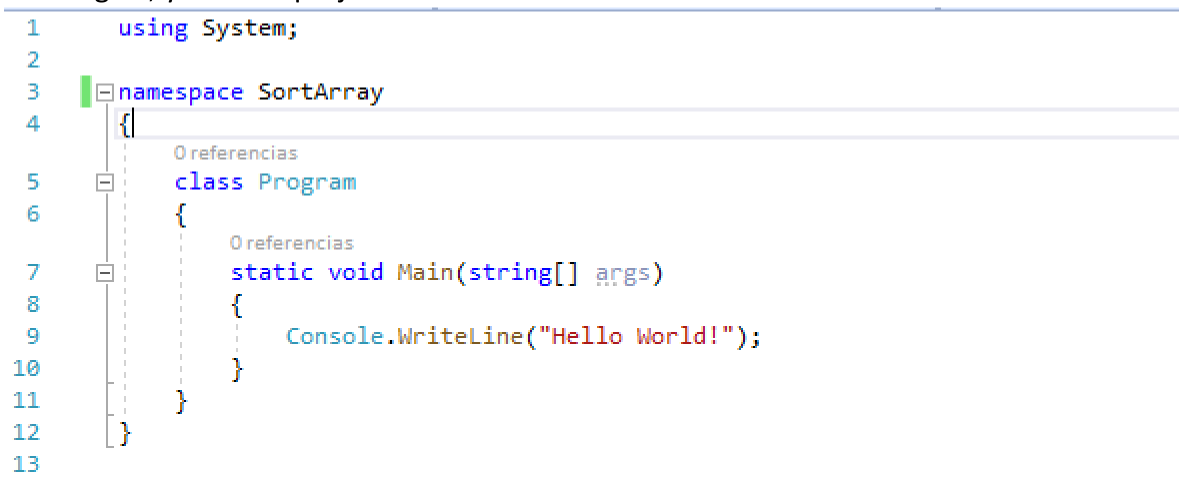
```
using System;

namespace ArrayExample
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] array = new int[5] { 1, 4, 2, 3, 5 };
            Console.WriteLine("---Default Array Elements---");
            foreach (int i in array)
            {
                Console.WriteLine(i);
            }
                Array.Sort(array);
                Console.WriteLine("---Elements After Sort---");
                foreach (int i in array)
                {
                    Console.WriteLine(i);
                }
                Array.Reverse(array);
                Console.WriteLine("---Elements After Reverse---");
                foreach (int i in array)
                {
                    Console.WriteLine(i);
                }
                Console.WriteLine("Press Enter Key to Exit..");
                Console.ReadLine();
        }
    }
}
```

11. You can compare this code with the one you just wrote and give it a try.
12. Save and close your Visual Studio project and get ready for the next lab!

Exercise 02: Read user inputs and print input in Array.

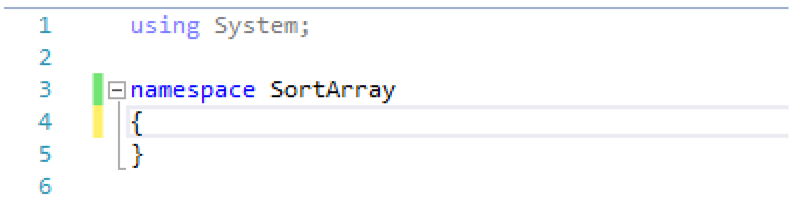
In this exercise we´ll request our users to input 10 numbers and the app will print them in the order that they were inputted.

1. Open Visual Studio 2019.
2. Create a **new project** choosing the **same settings we used in the 1st module** and change the name of the project to something you like (I named it `SortArray`).
3. Once again, your new project should look like this:



```
1       using System;
2
3      namespace SortArray
4       {
            0 referencias
5           class Program
6           {
                0 referencias
7               static void Main(string[] args)
8               {
9                   Console.WriteLine("Hello World!");
10              }
11          }
12      }
13
```

4. Delete lines 5 -11. Your screen should now look like this:



```
1       using System;
2
3      namespace SortArray
4       {
5       }
6
```

5. We´ll begin by creating a new array integer type and defining the number of elements in this array.

```
int[] arr = new int[10];
int i;
```

6. Now, well write some text, to let our user´s know what they need to do:

```
Console.Write("\n\ Module02: Sort items into an array\n");
Console.Write("-----------------------------------------\n");
Console.Write("Input 10 elements in the array :\n");
```

7. Create a `for` loop that will **ask the user for a new input**. We also want this instruction to be executed no more than 10 times since our **array is defined for 10 elements only**.

```
for (i = 0; i < 10; i++)

{

       Console.Write("element - {0} : ", i);

       arr[i] = Convert.ToInt32(Console.ReadLine());
}
```

8. Once the first loop finished executing, we´ll have to create a new loop that will now print the values in the order the users inputted them. This instruction will have to be looped 10 times too, since this array has 10 elements only.

```
Console.Write("\nElements in array are: ");
for (i = 0; i < 10; i++)
{
  Console.Write("{0}  ", arr[i]);
}
Console.Write("\n");
```

9. In the end, your final code should look like this:

```
using System;

namespace SortArray
{
    public class Exercise1
    {
        public static void Main()
        {
            int[] arr = new int[10];
            int i;
            Console.Write("\n\nModule02: Sort items into an array\n");
            Console.Write("-----------------------------------------\n");

            Console.Write("Input 10 elements in the array :\n");
            for (i = 0; i < 10; i++)
            {
                Console.Write("element - {0} : ", i);
                arr[i] = Convert.ToInt32(Console.ReadLine());
            }

            Console.Write("\nElements in array are: ");
            for (i = 0; i < 10; i++)
            {
                Console.Write("{0}  ", arr[i]);
            }
            Console.Write("\n");
        }
    }
}
```

10. You can now save your and close this project and and get ready for the next challenge.

Challenge

Create an array containing names that can display the whole list of names and the 2nd name in your array.

Module 03: ArrayLists.

Lab Setup

Estimated time: **50-60 min.**

Introduction

Now that we have a better understanding of how Arrays work in C#, it´s now time to explore ArrayLists and understand how useful they come when in need of arranging different types of elements in one single array.

Objectives

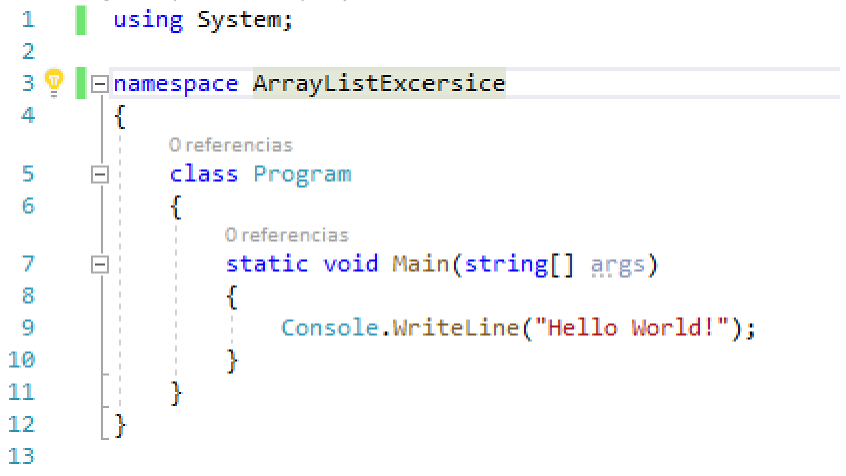
Very likely to Arrays, we have a similar way to arrange information called ArrayLists, in this module the student will be able to:

a. Learn when to use Arrays vs ArrayLists.
b. Create ArrayLists and manage information within them.

Exercise 01: Use ArrayLists to handle different type of elements.

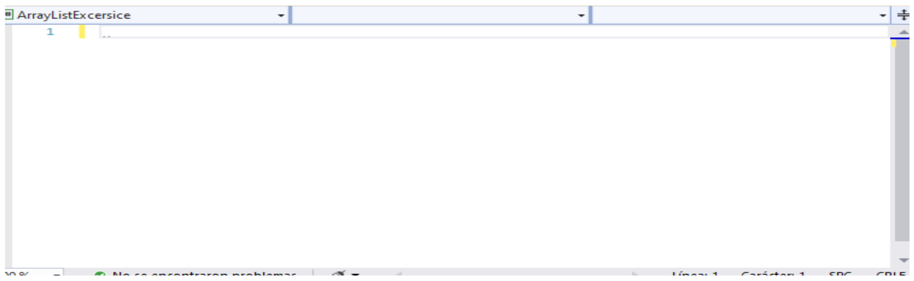
ArrayLists are a very useful tool when you need to handle different kinds of elements. In this exercise we´ll learn how we can create an ArrayLists and how to display/print this information into your console.

a. Open Visual Studio 2019.
b. Create a **new project** choosing the **same settings we used in the 1st module** and change the name of the project to something you like (I named it `ArrayListExcercise`).
c. Once again, your new project should look like this:

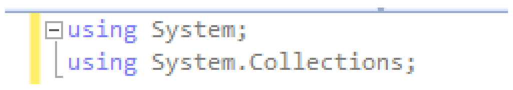


```
using System;

namespace ArrayListExcersice
{
    0 referencias
    class Program
    {
        0 referencias
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

d. Please delete the entire text so we can start coding in a blank workspace:



e. First of all we´ll need to declare the system and system collections (needed for the ArrayList to be readable) that we´ll be using for this exercise:



```
using System;
using System.Collections;
```

f. We´ll now create a new class and a new static void, and here is where fun begins, inside this bracket is where the rest of the magic will happen, so far, your code should be looking like this:

```
using System;
using System.Collections;
public class SamplesArrayList

{
    public static void Main()

    {

    }
}
```

g. We´re now all set to **start creating our ArrayList**, we´ll begin by naming our new ArrayList, and then we can add some info to the ArrayList, remember to write this inside the brackets of your `public static void Main()` :

```
ArrayList myAL = new ArrayList();
myAL.Add("Hello");
myAL.Add("Some korean phrase");
myAL.Add("Numb3r5 and letters");
```

h. We now want our app to display the properties and contents of our ArrayList. **To display the properties** of our ArrayList we´ll use the **“`Count`”** and **“`Capacity`”** commands in C#. We´ll write this code right below the one shown in step 7:

```
Console.WriteLine("My ArrangeList");
Console.WriteLine("    Count:    {0}", myAL.Count);
Console.WriteLine("    Capacity: {0}", myAL.Capacity);
Console.Write("    Values:");
PrintValues(myAL);
```

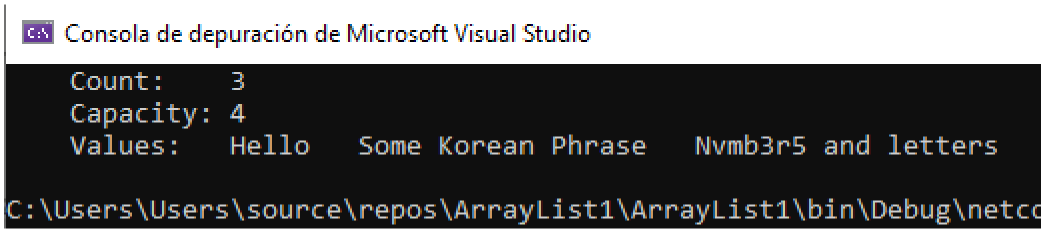
i. At this point the `public static void Main()` brackets can be closed and we´ll create a new `public static void` that we´ll be using to call **`PrintValues`** and display the information contained in our ArrayList:

```
public static void PrintValues(IEnumerable myList)
    {
        foreach (Object obj in myList)
            Console.Write("   {0}", obj);
        Console.WriteLine();
    }
```

j. In the end, your final code should look like this:

```
using System;
using System.Collections;
public class SamplesArrayList
{

    public static void Main()
    {

        // Creates and initializes a new ArrayList.
        ArrayList myAL = new ArrayList();
        myAL.Add("Hello");
        myAL.Add("Some Korean Phrase");
        myAL.Add("Nvmb3r5 and letters");

        // Displays the properties and values of the ArrayList.
        Console.WriteLine("My ArrangeList");
        Console.WriteLine("    Count:    {0}", myAL.Count);
        Console.WriteLine("    Capacity: {0}", myAL.Capacity);
        Console.Write("    Values:");
        PrintValues(myAL);
    }

    public static void PrintValues(IEnumerable myList)
    {
        foreach (Object obj in myList)
            Console.Write("   {0}", obj);
        Console.WriteLine();
    }
   }
```

k. Now it´s time for you to compile the app and test the results that the console will be displaying:


Consola de depuración de Microsoft Visual Studio

```
   Count:    3
   Capacity: 4
   Values:   Hello   Some Korean Phrase   Nvmb3r5 and letters

C:\Users\Users\source\repos\ArrayList1\ArrayList1\bin\Debug\netc
```

Summary

In this module, we learned about the properties of an ArrayList, and how an they handle different types of elements, while arrays are only for elements of the same kind.

Challenge

Create an app that allows you to insert an element in your array.

Module 04: Introduction to classes.

Lab Setup

Estimated time: **35 min.**

Introduction

During your course, you learned that the fundamentals of OOP are based on classes and objects. Classes are user-defined blueprints that you use as templates that you use to create objects that consist on: State, behavior and identity.

Object Oriented Programming (OOP) is mainly used to develop things like videogames, apps, GUIs and tons of virtual solutions more. In this lab exercises, you´ll put in practice all theory and concepts learned during this course.

Objectives

As we saw in the previous class, constructors are another important concept used in C#, in this lab session we´ll put in practice the concept learned in class. After this lab exercises, you´ll be able to:

a. Understand funds in constructors.
b. Learn how to initialize your constructor when you first compile your app.
c. Get input from the user and sort it in an array.
d. Display the newly added info to the console.

Exercise 01: Constructors

In this exercise, we´ll create a simple app that will ask the user to input 3 names and it will store them in an Array and them print them in the console.

1. Open Visual Studio 2019 and create a new project named constructors.
2. Once you have created your project, please delete all pre-loaded info so we can start in a blank page.
3. We´ll begin by copy/pasting the following code, which for now will only print the instructions to this lab.

```
4. using System;

5. public class Constructors
6. {
                {
                     Console.WriteLine("Please write a name and hit enter key");
                }

7. }
```

8. Once we have this piece of code in our sheet, then we´ll begin by creating a Person class that has a name property of type string. Please copy and paste the following code into your `public class Constructors`.

```
public static void Main(string[] args)

{

     int total = 3;

     Person[] persons = new Person[total];

     for (int i = 0; i < total; i++)

     {

            persons[i] = new Person(Console.ReadLine());
```

```
        }
            for (int i = 0; i < total; i++)
        {
            Console.WriteLine(persons[i].ToString());
        }
    }
```

9. What we did here is to create an object that contains an integer named total, and a string-type array named `Person,` this array will contain 3 elements that will be inputted by the user.
10. In this step we will be creating a new constructor that will receive the name as a parameter. Please copy and paste the following code right below the last piece of code you wrote in your app.

```
    public class Person
    {
        public string Name { get; set; }
        public Person(string name)
        {
            Name = name;
        }

        public override string ToString()
        {
            return "Hello! My name is " + Name;
        }
        ~Person()
        {
            Name = string.Empty;
```

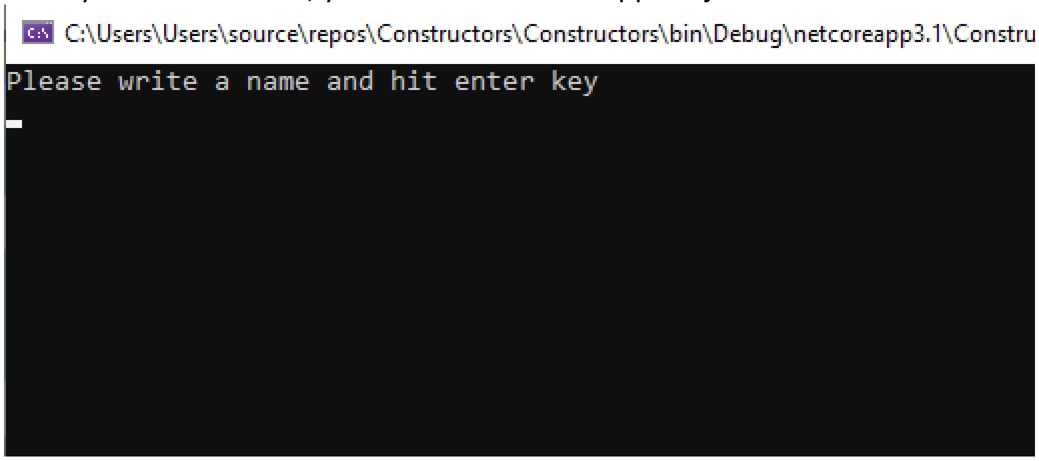
```
                }

            }

            }
```

11. In this last part of the code, we´re also creating a destructor that assigns the `Name` to our `Empty;` string.
12. At the end your entire code should look like this:

```
using System;

public class Constructors
{

public static void Main(string[] args)
{
        {
                Console.WriteLine("Please write a name and hit enter
                key");
        }

        int total = 3;
        Person[] persons = new Person[total];

        for (int i = 0; i < total; i++)
        {
                persons[i] = new Person(Console.ReadLine());
        }

        for (int i = 0; i < total; i++)
        {
                Console.WriteLine(persons[i].ToString());
        }
}


public class Person
{
        public string Name { get; set; }

            public Person(string name)
            {
                    Name = name;
            }

            public override string ToString()
            {
                    return "Hello! My name is " + Name;
            }

            ~Person()
            {
```

```
                        Name = string.Empty;
                    }
                }

                }
```

13. Once you have finished, you can now test the app we just created:

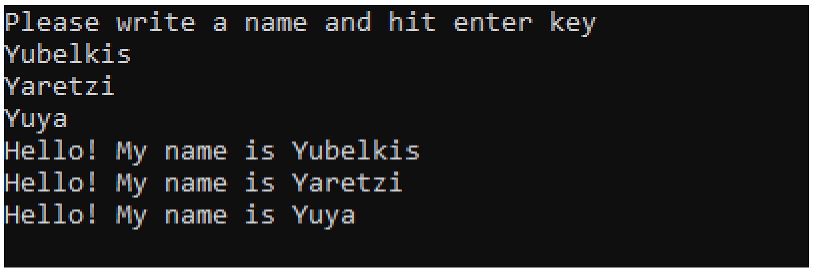


C:\Users\Users\source\repos\Constructors\Constructors\bin\Debug\netcoreapp3.1\Constru

```
Please write a name and hit enter key
```

14.

15. And once you input the names you want, the app will display them in the console:



```
Please write a name and hit enter key
Yubelkis
Yaretzi
Yuya
Hello! My name is Yubelkis
Hello! My name is Yaretzi
Hello! My name is Yuya
```

16. Save and close your Visual Studio Project.

Module 05: Inheritance & Polymorphism.

Lab Setup

Estimated time: **90 min.**

Introduction

In the previous module, we learned some of the pilar features of OOP like how to create objects and polymorphism, that will let us access elements in parent classes from their child classes and let us use those elements from the parent class and manage them from the child class.

I know this might sound a little confusing, but no worries, this will make much more sense once we make some lab exercises to clarify all the theory mentioned before.

Objectives

This module will be great for the student to understand the basic logic used when coding in C#, some of the most relevant concepts that we´ll be practicing during this module are:

a. Creating classes with public values.
b. Creating parent and child classes.
c. Using child classes to read information from the parent class.
d. Display this information to the console.
e. Understand polymorphism and it´s usage.

Exercise 01: Inheritance of objects. Base and Derived classes.

In this exercise, we´ll create a simple app that will demonstrate how inheritance works in C#.

1. Open Visual Studio 2019 and create a new project named Inheritance.
2. Once you have created your project, please delete all pre-loaded info so we can start in a blank page.
3. This topic is a little hard to comprehend, so let´s start with this example that will help us understand how inheritance works. Let´s begin by creating our parent class which will be named A:

```
using System;
namespace Inheritance
{
	class A
	{
		public int a;
		public A()
		{
			a = 5;
		}
	}
}
```

4. We´ll now create a new class with that will contain a variable and a constructor, please write or copy/paste the following instructions:

```
class B : A
{
        public int b;
        {
                b = 20;
        }
        public int getSum()
        {
                return a + b;
        }
}
```

5. What we did above was to create a constructor which will initialize the value of `b` to 20. Below that we created a method that returns the sum of a and b.

6. And to finish this app, we´ll now create our last class, name it Run for this activity, and this class will contain our main method.

```
class Run
{
        static void Main(string[] args)
        {
                B obj = new B();
                Console.WriteLine(obj.getSum());
        }
}
```

7. The main method is used to initialize the class B object, and it will also print the result of the getSum method.
8. Your final code should be looking like this:

```
using System;
namespace InheritanceDemo
{
        class A

        {
                public int a;
                public A()
                {
                        a = 5;
                }
        }

        class B : A
```

```
            {
                    public int b;
                    public B()
                    {
                            b = 20;
                    }
                    public int getSum()
                    {
                            return a + b;
                    }
            }

            class Run

            {
                    static void Main(string[] args)
                    {
                            B obj = new B();
                            Console.WriteLine(obj.getSum());
                    }
            }
    }
```

Exercise 02: Hierarchical Inheritance

In the previous exercise we learned how we can create parent classes and how we can link them to a child class. In this lesson, we´ll test how multiple classes can inherit properties from a single base class.

1. Open Visual Studio 2019 and create a new project named Hinheritance.
2. Once you have created your project, please delete all pre-loaded info so we can start in a blank page.
3. Let´s begin by defining our workspace, please copy/paste the following code:

```
using System;
namespace Hinheritance
{
}
```

4. Just like in the previous exercise, we will create a parent class A, but in this scenario, there will be other two classes, named B and C.

```
class A
    {
        public int a;
        public A()
```

```
            {
                a = 10;
            }
        }

    class B : A
        {
            public int b;
            public B()
            {
                b = 7;
            }
        }

    class C : A
        {
            public int c;
            public C()
            {
                c = 23;
            }
               }
```

5. Just like in the first exercise of this module, we will need to create a test class for executing the app.

```
    {
            static void Main(string[] args)

            {
                Console.WriteLine("Hierarchical inheritance");
                B obj1 = new B();
                C obj2 = new C();
                Console.WriteLine("Using class B object (obj1)");
                Console.WriteLine("a = {0}", obj1.a);
                Console.WriteLine("b = {0}", obj1.b);

                Console.WriteLine("Using class C object (obj2)");
                Console.WriteLine("a = {0}", obj2.a);
                Console.WriteLine("c = {0}", obj2.c);
            }
               }
```

6. It´s important to pay attention to the highlighted code here we demonstrate what is happening.

   We´re able to access and print information contained in class A, from an object that is created in Class C.

7. Your final code should look like this:

```
using System;
namespace InheritanceDemo
{
    class A
    {
        public int a;
        public A()
        {
            a = 10;
        }
    }


    class B : A
    {
        public int b;
        public B()
        {
            b = 7;
        }
    }


 class C : A
    {
        public int c;
        public C()
        {
            c = 23;
        }
    }


class Test
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hierarchical inheritance");
            B obj1 = new B();
            C obj2 = new C();
            Console.WriteLine("Using class B object (obj1)");
            Console.WriteLine(obj1.a);
            Console.WriteLine(obj1.b);
            Console.WriteLine("Using class C object (obj2)");
            Console.WriteLine(obj2.a);
            Console.WriteLine(obj2.c);
        }
    }
}
```

8. Compile the program to test it and see the result.
9. Save and close your Visual Studio Project.

Exercise 03: Polymorphism
Now that we understand how inheritance works, we´ll show you another great and fundamental feature used when coding in C#.

1. Open Visual Studio 2019 and create a new project named Hinheritance.
2. Once you have created your project, please delete all pre-loaded info so we can start in a blank page.
3. Let´s begin by defining our workspace, please copy/paste the following code:

```
        using System;

        namespace PolyM
        {

        }
```

4. Once again, we´ll begin this exercise by defining a base class

```
class Animal
{
        public void animalSound()
        {
                Console.WriteLine("The animal makes a sound");
        }
}
```

5. Now that we have a base class, we´ll create two more derived classes.

```
        class Pig : Animal
        {
                public void animalSound()
                {
                        Console.WriteLine("The pig says: wee wee");
                }
        }

                class Dog : Animal
                {
                public void animalSound()
                {
                Console.WriteLine("The dog says: bow wow");
                }
        }
```

6. Finally, we´ll create the class that contain the instructions to run the program.

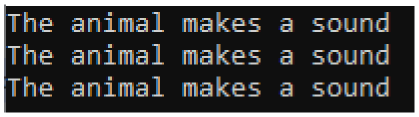
```
  class Program
    {
        static void Main(string[] args)
        {
            Animal myAnimal = new Animal();
            Animal myPig = new Pig();
            Animal myDog = new Dog();

            myAnimal.animalSound();
            myPig.animalSound();
            myDog.animalSound();
        }
    }
```

7. The complete code must look like this:

```
using System;

namespace PolyM
{
    class Animal
    {
        public void animalSound()
        {
            Console.WriteLine("The animal makes a sound");
        }
    }

    class Pig : Animal
    {
        public void animalSound()
        {
            Console.WriteLine("The pig says: wee wee");
        }
    }

    class Dog : Animal
    {
        public void animalSound()
        {
            Console.WriteLine("The dog says: bow wow");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Animal myAnimal = new Animal();
            Animal myPig = new Pig();
            Animal myDog = new Dog();

            myAnimal.animalSound();
            myPig.animalSound();
            myDog.animalSound();
        }
    }
}
```

8. Please run the program.

9. You might find nothing interesting in the result, since is not what we were expecting to do, right?

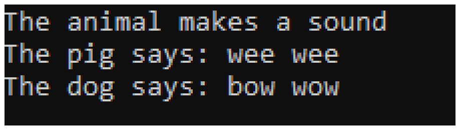


```
The animal makes a sound
The animal makes a sound
The animal makes a sound
```

10. This happens because the base class method overrides the derived methods, to solve this issue by adding the keyword virtual in the bass class and using override in each of the derived classes. Please notice how we modified the classes with the virtual and override tags:

```
class Animal
{
    public virtual void animalSound()
    {
        Console.WriteLine("The animal makes a sound");
    }
}

class Pig : Animal
{
    public override void animalSound()
    {
        Console.WriteLine("The pig says: wee wee");
    }
}

class Dog : Animal
{
    public override void animalSound()
    {
        Console.WriteLine("The dog says: bow wow");
    }
}
```

11. Please just add the virtual and override methods to your classes, execute the program and see how it works now. It should be showing you the output we were expecting from the beginning.



```
The animal makes a sound
The pig says: wee wee
The dog says: bow wow
```

12. Save and close the project.

Conclusion

In this module we learned how classes and it´s properties can be inherited to other classes, and how useful this way of accessing information is, not only because it shortens data, but also because it allows you to reuse fields and methods of an existing class.

Module 05: FILE I/O

Lab Setup

Estimated time: **30-40 min.**

Introduction

In class, you learned that arrays in C# are meant to store multiple variables of the same type. First you declare an array by specifying the type of it´s elements. It´s important to remark that Arrays can be single-dimensional, multidimensional or jagged and that an array with n elements is indexed from 0 to n-1.

Objectives

In this module, the student will be able to:

a. Access data in your local files.
b. Be able to read .txt filles.
c. Display the information of that file in the console.
d. Create files from the console to the local storage.
e. Input text from the app to the newly created .txt file

Exercise 01: Reading files with C#

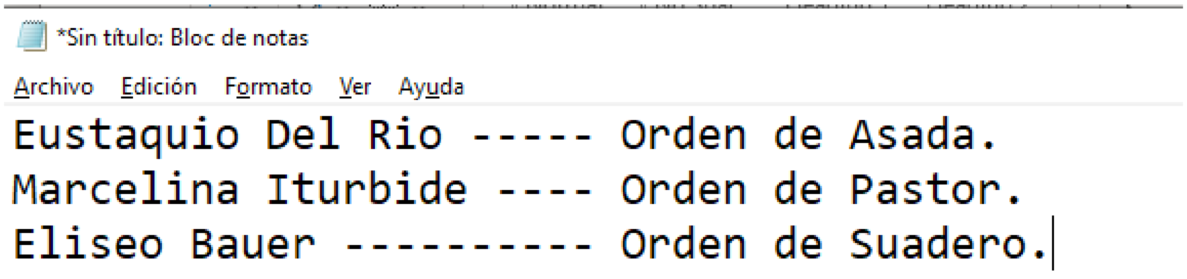
In this exercise, we´ll demonstrate how to read the information contained in a .txt file using C#.

1. Open Visual Studio 2019 and create a new project named ReadFile.
2. Please delete the pre-loaded code in the project and copy/paste the following one:

```
using System;
using System.IO;

class ReadFile
```

```
{
    public static void Main()
    {

    }
}
```

3. Now, open your notepad in windows, and create a new file and type in anything that comes to your mind, in my scenario I´ll be reading the To Go orders from a famous Mexican restaurant, that´s because you can read the taco each customer ordered.



```
Eustaquio Del Rio ----- Orden de Asada.
Marcelina Iturbide ---- Orden de Pastor.
Eliseo Bauer ---------- Orden de Suadero.
```

4. Once your satisfied with your .txt file, please save it to this path `c:\temp\` (you can save it in any other path, please just make sure you remember it) in my case I named my file ToGo, since this is the To Go list order from the restaurant.

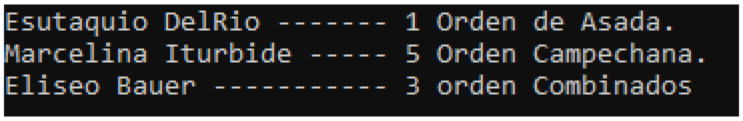
5. After you saved your file, please close it and let´s get back to the code.
6. Now we´re going to declare the path where our file is allocated Inside the Main(), we´ll do this by declaring the following:

```
string path = @"c:\temp\ToGo.txt";
```

7. And now, we will ask the application to open the file so we can read the information contained in there.

```
using (StreamReader sr = File.OpenText(path))
 {
     string s;
     while ((s = sr.ReadLine()) != null)
     {
         Console.WriteLine(s);
     }
 }
```

8. Run the code and see how the console is displaying the information contained in the .txt file we create earlier in this lab.



```
Esutaquio DelRio ------- 1 Orden de Asada.
Marcelina Iturbide ----- 5 Orden Campechana.
Eliseo Bauer ----------- 3 orden Combinados
```

9. Please don´t close your project yet, we´ll explore more on the files class in the next lab!

10. If the project didn´t compile right, or something went wrong, please take a look at the full code in this exercise:

```csharp
using System;
using System.IO;

class ReadFile
{
    public static void Main()
    {
        string path = @"c:\temp\ToGo.txt";

        using (StreamReader sr = File.OpenText(path))
        {
            string s;
            while ((s = sr.ReadLine()) != null)
            {
                Console.WriteLine(s);
            }
        }
    }
}
```

Exercise 02: Creating files and writing lines into them in C#

In the last exercise we demonstrated how you can read files allocated in your device. In this lab, we´ll make our app to check if the files already exists, if so, it will just read it but if it doesn´t, then it will create a new file.

1. Using the code from the previous lab:

```
using System;
using System.IO;

class ReadFile
{
    public static void Main()
    {
        string path = @"c:\temp\ToGo.txt";


        using (StreamReader sr = File.OpenText(path))
        {
            string s;
            while ((s = sr.ReadLine()) != null)
            {
                Console.WriteLine(s);
            }
        }
    }
}
```

2. In this step, we´ll request the app to verify if the file already exist, please write this part of the code where the red rectangle shows above:

```
if (!File.Exists(path))
        {
            // Create a file to write to.
            using (StreamWriter sw = File.CreateText(path))
            {
                sw.WriteLine("No");
                sw.WriteLine("More");
                sw.WriteLine("Tacos");
            }
        }
```

3. What we´re doing above is to use an if conditional to check if the path exists, and if not, then to create a file in the path that we previously stated.
4. Now go back to the path where the .txt file is and delete it!

   This way you´ll be able to demonstrate that we´re writing a new file in the path we specified. Make sure your code looks like this:

```
using System;
using System.IO;

class ReadFile
{
    public static void Main()
    {
        string path = @"c:\temp\ToGo.txt";
        if (!File.Exists(path))
        {
            using (StreamWriter sw = File.CreateText(path))
            {
                sw.WriteLine("Hello");
                sw.WriteLine("And");
                sw.WriteLine("Welcome");
            }
        }

        using (StreamReader sr = File.OpenText(path))
        {
            string s;
            while ((s = sr.ReadLine()) != null)
            {
                Console.WriteLine(s);
            }
        }
    }
}
```

Module 06: Introduction to LinQ query

Lab Setup

Estimated time: **30 min.**

Introduction

In class, you learned that arrays in C# are meant to store multiple variables of the same type. First you declare an array by specifying the type of it´s elements. It´s important to remark that Arrays can be single-dimensional, multidimensional or jagged and that an array with n elements is indexed from 0 to n-1.

Objectives

This lab exercise will help you get familiar with the syntax of LinQ when integrating it with C# and also very important he will learn to call the system needed for Linq utilization.

Exercise 01: Writing your first LinQ query with C#

We will create an array that will contain multiple integers, all integers greater than 5 will be returned and displayed in the console.

I. Open Visual Studio 2019 and create a new project and name it LinQ.
II. Please delete all pre-loaded code in the project.
III. We´ll begin our code by declaring the System needed to use the LinQ system:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

IV. Notice how it is important to declare that we´ll be using the linq system since the very beginning.
V. Once this said let´s now start coding our workspace as usual, just like we have done in the previous labs:

```
namespace LinQ
{
    class Program
    {
       static void Main(string[] args)
              {

               }
    }

}
```

VI. So now we have declared our main method we´ll continue by creating an integer list inside our main method, this method will contain an integer list made of 10 numbers from 1-10.

```
List<int> integerList = new List<int>()
{
        1, 2, 3, 4, 5, 6, 7, 8, 9, 10
};
```

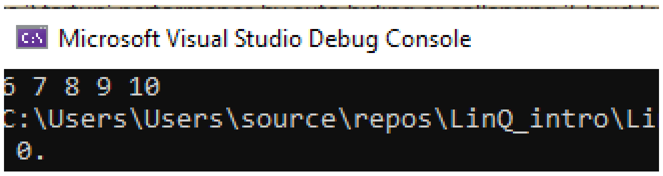
VII. It´s now the time to use LinQ and query syntax for the first time!

```
var QuerySyntax = from obj in integerList
where obj > 5
select obj;
```

VIII. Finally, in the execution part we´ll make our results to be displayed by the program in our console. Don’t forget to close your main method after this last part.

```
foreach (var item in QuerySyntax)
{
        Console.Write(item + " ");
}
        Console.ReadKey();
```

IX. Finally, in the execution part we´ll make our results to be displayed by the program. Please run your app and see the result.



Microsoft Visual Studio Debug Console

```
6 7 8 9 10
C:\Users\Users\source\repos\LinQ_intro\Li
 0.
```

X. Please save and close your project.