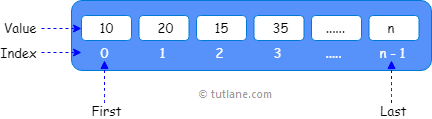
# C# Arrays with Examples

In c#, **Arrays** are useful to store multiple elements of the same data type at contiguous memory locations and arrays will allow us to store a fixed number of elements sequentially based on the predefined number of items.

In the previous chapter, we learned about [variables in c#](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples), which will help us to hold a single value like **int x = 10;**. In case if we want to hold more than one value of the same [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples), then an arrays came into the picture in c# to solve this problem.

An array can start storing the values from index **0**. Suppose if we have an array with n elements, then it will start storing the elements from index **0** to **n-1**.

Following is the pictorial representation of storing the multiple values of the same type in the c# array data structure.



If you observe the above diagram, we are storing the values in an array starting from index **0** and it will continue to store the values based on the defined number of elements.

## C# Arrays Declaration

In c#, **Arrays** can be declared by specifying the type of elements followed by the square brackets **[]** like as shown below.

type[] array\_name;

Here the **type** is nothing but a [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) of elements to store in an array and **array\_name** represents the name of an array.

For example, the following are the different ways of declaring an array with different [data types](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) in c# programming language.

// Store only int values

int[] numbers;

//Store only string values

string[] names;

//Store only double values

double[] ranges;

If you observe the above examples, we declared arrays with the required data type based on our requirements.

In c#, the array elements can be of any type and by default, the values of numeric array elements are set to zero and the reference elements are set to null.

## C# Arrays Initialization

In c#, Arrays can be initialized by creating an instance of an array with a **new** keyword. By using a **new** keyword we can declare and initialize an array at the same time based on our requirements.

Following are the different ways of declaring and initializing array elements by using the **new** keyword in c# programming language.

// Declaring and Initializing an array with size of 5

int[] array = new int[5];

//Defining and assigning an elements at the same time

int[] array2 = new int[5]{1,2,3,4,5};

//Initialize with 5 elements will indicates the size of an array

int[] array3 = new int[] { 1, 2, 3, 4, 5 };

// Another way to initialize an array without size

int[] array4 = { 1, 2, 3, 4, 5 };

// Declare an array without initialization

int[] array5;

array5 = new int[]{ 1, 2, 3, 4, 5 };

If you observe above examples, in first statement we declared and initialized an integer array with the size of **5** to allow an array to store **5** integer values and the array can contain an elements from **array[0]** to **array[4]**.

In the second statement, we declared and initialized an array same as the first statement but also assigned values to each index followed by curly brackets **{ }**.

In a third or fourth statement, while declaration we initialized an array with values, but without specifying any size. Here, the size of an array can be determined by the number of elements so the size initializer is not required if we are assigning elements during the initialization.

In c#, we can declare an array variable without initialization, but we must use the **new** keyword to assign an array to the variable.

In the fifth statement, we declared an array without initialization and we used a **new** keyword to assign array values to the variable.

In c#, after an array declaration, we can initialize array elements using index values. Following is an example of declaring and initializing array elements using individual index values in c#.

int[] array = new int[5];

array[0] = 1;

array[1] = 2;

array[2] = 3;

array[3] = 4;

array[4] = 5;

If you observe above example, we are initializing an array elements individual using individual index values.

Generally, in c# initializing an array without **size** or assigning values to an array without a **new** operator will throw compile-time errors. For example:

// Error. Initialize an array without size

int[] array = new int[];

// Error. Initialize an array without new keyword

int[] array1;

array1 = { 1, 2, 3, 4, 5 };

If you observe the above examples, in the first statement we initialized an array without any **size** and in the second statement, we declared and initializing array elements without using the **new** keyword. These two statements will throw compile-time errors in our c# applications.

## C# Accessing an Array Elements

In c#, we can access array elements by using [for loop](http://tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) or [foreach loop](http://tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "C# Foreach Loop with Examples" \t "_blank) or with particular index numbers.

Following is the code snippet of accessing array elements by using particular index numbers.

int[] array = new int[5] { 1, 2, 3, 4, 5 };

int a = array[1]; // It will return 2

int b = array[4]; // It will return 5

If you observe above code, we are trying to access an array elements using index values in c#.

Following is the example of declaring, initializing and accessing array elements with particular index numbers in c# programming language.

using System;

namespace YourApp

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] array = new int[5] { 1, 2, 3, 4, 5 };

            Console.WriteLine(array[0]);

            Console.WriteLine(array[1]);

            Console.WriteLine(array[2]);

            Console.WriteLine(array[3]);

            Console.WriteLine(array[4]);

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

            Console.ReadLine();

        }

    }

}

If you observe above example, we declared and initialized an array with 5 elements and we are accessing an array elements using index values.

Try executing the above program.

If you observe the above result, we are able to access array elements using index numbers based on our requirements.

## C# Access Array Elements with For Loop

In c#, by using [for loop](http://tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) we can iterate through array elements and access the values of an array with length property.

Following is the example of accessing array elements using for loop in c# programming language.

using System;

namespace YourApp

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] array = new int[5] { 1, 2, 3, 4, 5 };

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

            {

                Console.WriteLine(array[i]);

            }

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

            Console.ReadLine();

        }

    }

}

If you observe above example, we looping through an array elements with [for loop](http://tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) to access an array elements based on our requirements.

Try executing the above program

If you observe the above result, we are able to loop through elements in an array with [for loop](http://tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) and print array values based on our requirements.

## C# Access Array Elements with Foreach Loop

In c#, same as [for loop](http://tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) we can use the [foreach loop](http://tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "C# Foreach Loop with Examples" \t "_blank) to iterate through array elements and access the values of an array based on our requirements.

Following is the example of accessing array elements using a [foreach loop](http://tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "C# Foreach Loop with Examples" \t "_blank) in c# programming language.

using System;

namespace YourApp

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] array = new int[5] { 1, 2, 3, 4, 5 };

            foreach(int i in array)

            {

                Console.WriteLine(i);

            }

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

            Console.ReadLine();

        }

    }

}

If you observe above example, we looping through an array elements with [foreach loop](http://tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "C# Foreach Loop with Examples" \t "_blank) to access an array elements based on our requirements.

Try executing the program.

If you observe the above result, we are able to loop through elements in an array with [foreach loop](http://tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "C# Foreach Loop with Examples" \t "_blank) and print array values based on our requirements.

This is how we can access array elements in c# programming language based on our requirements.

# C# List (List<T>)

In c#, **List** is a generic type of collection so it will allow storing only strongly typed objects i.e. elements of the same [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) and the size of the list will vary dynamically based on our application requirements like adding or removing elements from the list.

In c#, the list is same as an [ArrayList](https://www.tutlane.com/tutorial/csharp/csharp-arraylist" \o "ArrayList in C# with Examples" \t "_blank) but the only difference is [ArrayList](https://www.tutlane.com/tutorial/csharp/csharp-arraylist" \o "ArrayList in C# with Examples" \t "_blank) is a non-generic type of [collection](https://www.tutlane.com/tutorial/csharp/csharp-collections) so it will allow storing elements of different [data types](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples).

## C# List Declaration

In c#, the list is a generic type of [collection](https://www.tutlane.com/tutorial/csharp/csharp-collections) and it is provided by **System.Collections.Generic** namespace.

As discussed, the [collection](https://www.tutlane.com/tutorial/csharp/csharp-collections) is a [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) so to define a list, you need to declare an instance of the list class before we perform any operations like add, delete, etc. like as shown below.

List<T> lst = new List<T>();

If you observe the above list declaration, we created a generic list (**lst**) with an instance of list class using type parameter (**T**) as a placeholder with an angle (**<>**) brackets.

Here, the angle (**<>**) brackets will indicate that the list is a **generic** type and type parameter (**T**) is used to represent the type of elements to be accepted by the list.

In c#, the generic list (**List<T>**) is an implementation of **IList<T>** [interface](https://www.tutlane.com/tutorial/csharp/csharp-interface) so we can also use **IList<T>** [interface](https://www.tutlane.com/tutorial/csharp/csharp-interface) to create an object of the generic list (**List<T>**) like as shown below.

IList<T> lst = new List<T>();

## C# List Initialization

Following is the example of initializing a generic list by specifying a required type to accept values to store.

List<string> lst = new List<string>();

If you observe above example, we defined a list (**lst**) with **string** [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) to store only string elements.

## C# List Properties

The following are some of the commonly used [properties](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set) of the generic list in c# programming language.

| **Property** | **Description** |
| --- | --- |
| Capacity | It is used to get or set the number of elements a list can contain. |
| Count | It is used to get the number of elements in the list. |
| Item | It is used get or set an element at the specified index. |

## C# List Methods

The following are some of the commonly used [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) of the generic list to perform add, search, insert, delete or sort operations in c# programming language.

| **Method** | **Description** |
| --- | --- |
| Add | It is used to add an element at the end of the List. |
| AddRange | It is used to add all the elements of the specified collection at the end of the List. |
| Clear | It will remove all the elements from the List. |
| Contains | It is used determine whether the specified element exists in the List or not. |
| CopyTo | It is used to copy the entire List to a compatible one-dimensional array. |
| Find | It is used to search for an element that matches the conditions defined by the specified predicate and returns the first occurrence of the List. |
| FindAll | It is used to retrieve all the elements that match the conditions defined by the specified predicate. |
| ForEach | It is used to iterate through the List to access elements. |
| Insert | It is used to insert an element into the List at the specified index. |
| InsertRange | It is used to insert all the elements of the specified collection into List starting from the specified index. |
| Remove | It is used to remove the first occurrence of a specified element from the List. |
| RemoveAt | It is used to remove an element from the List based on the specified index position. |
| RemoveRange | It is used to remove a range of elements from the List. |
| Reverse | It reverses the order of List elements. |
| Sort | It sorts the elements in the List. |
| ToArray | It will copy the elements of the List to new array objects. |

## C# Generic List (List<T>) Example

Following is the example of using the generic list (**List<T>**) in the c# programming language.

using System;

using System.Collections.Generic;

namespace YourA

{

    class Program

    {

        static void Main(string[] args)

        {

            // Creating and initializing list

            List<int> lst = new List<int>();

            lst.Add(1);

            lst.Add(8);

            lst.Add(45);

            List<string> lst2 = new List<string>();

            lst2.Add("Hi");

            lst2.Add("Welcome");

            lst2.Add("to");

            lst2.Add("YourApp");

            Console.WriteLine("List1 Elements Count: " + lst.Count);

            Console.WriteLine("List1 Capacity: " + lst.Capacity);

            Console.WriteLine("\*\*\*\*\*\*\*\*\*List1 Elements\*\*\*\*\*\*\*\*");

            // Accessing list elements

            foreach (var item in lst)

            {

                Console.WriteLine(item);

            }

            Console.WriteLine("List2 Elements Count: " + lst2.Count);

            Console.WriteLine("List2 Capacity: " + lst2.Capacity);

            Console.WriteLine("\*\*\*\*\*\*\*\*\*List2 Elements\*\*\*\*\*\*\*\*");

            foreach (var item in lst2)

            {

                Console.WriteLine(item);

            }

            Console.ReadLine();

        }

    }

}

If you observe above example, we are able to define a new generic lists (**lst**, **lst2**) collection by using **System.Collections.Generic**  [namespace](https://www.tutlane.com/tutorial/csharp/csharp-namespaces-with-examples" \o "Namespaces in C# with Examples" \t "_blank). Here, we added only the defined [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) (**string**) values to the newly created lists (**lst**, **lst2**) by using **Add** method and accessing those generic list (**lst**, **lst2**) elements by using [foreach](https://www.tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "Foreach Loop in C# with Examples" \t "_blank) loop.

Try executing the above program.

This is how you can use a generic list in c# to store the group of defined type elements based on our requirements.

The following are some of the useful examples to work with a generic list in c# programming language.

## C# Add Elements to List

In c#, you can add elements to the list either by using **Add** / **AddRange** methods or at the time of initialization based on our requirements.

Following is the example of adding elements to the list using **Add** or **AddRange** methods in c#.

using System;

using System.Collections.Generic;

namespace YourApp

{

    public class User

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public string Location { get; set; }

    }

    class Program

    {

        static void Main(string[] args)

        {

            // Creating and initializing list

            List<int> lst = new List<int>() { 1, 8, 45, 70 };

            Console.WriteLine("List1 Elements Count: " + lst.Count);

            Console.WriteLine("\*\*\*\*\*\*\*\*\*List1 Elements\*\*\*\*\*\*\*\*");

            // Accessing list elements

            foreach (var item in lst)

            {

                Console.WriteLine(item);

            }

            //AddRange method

            List<int> lst2 = new List<int>();

            lst2.AddRange(lst);

            Console.WriteLine("List2 Elements Count: " + lst2.Count);

            Console.WriteLine("\*\*\*\*\*\*\*\*\*List2 Elements\*\*\*\*\*\*\*\*");

            foreach (var item in lst2)

            {

                Console.WriteLine(item);

            }

            List<User> users = new List<User>(){

            new User{Id= 1, Name= "Suresh Dasari", Location= "Hyderabad"},

            new User{Id = 2, Name = "Rohini Alavala", Location = "Guntur"}

            };

            users.Add(new User { Id = 3, Name = "Trishika Dasari", Location = "Guntur" });

            users.Add(new User { Id = 4, Name = "Praveen Alavala", Location = "Eluru" });

            Console.WriteLine("List3 Elements Count: " + users.Count);

            Console.WriteLine("\*\*\*\*\*\*\*\*\*List3 Elements\*\*\*\*\*\*\*\*");

            foreach (User u in users)

            {

                Console.WriteLine("Id:{0}, Name:{1}, Location:{2}", u.Id, u.Name, u.Location);

            }

            Console.ReadLine();

        }

    }

}

If you observe above example, we created a multiple lists (**lst**, **lst2**, **users**) and adding an elements to the lists in different ways like added some of the elements during initialization time and some other elements by using **Add** & **AddRange** methods.

Try executing the above program.

## C# Access List Elements

In c#, we have different ways to access list elements i.e. either by using index positions or by iterating through the list using [for](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) / [foreach](https://www.tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "Foreach Loop in C# with Examples" \t "_blank) loops.

Following is the example of accessing list elements in different ways.

using System;

using System.Collections.Generic;

namespace YourApp

{

    public class User

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public string Location { get; set; }

    }

    class Program

    {

        static void Main(string[] args)

        {

            // Creating and initializing list

            List<int> lst = new List<int>() { 1, 8, 45, 70 };

            Console.WriteLine("\*\*\*\*\*\*\*\*\*Access Elements with Index Position\*\*\*\*\*\*\*\*");

            Console.WriteLine("Element at 0: " + lst[0]);

            Console.WriteLine("Element at 2: " + lst[2]);

            // Creating List

            List<User> users = new List<User>(){

            new User{Id= 1, Name= "Suresh Dasari", Location= "Hyderabad"},

            new User{Id = 2, Name = "Rohini Alavala", Location = "Guntur"},

            new User { Id = 3, Name = "Trishika Dasari", Location = "Guntur" },

            new User { Id = 4, Name = "Praveen Alavala", Location = "Eluru" }

            };

            Console.WriteLine("\*\*\*\*\*\*\*\*\*Access Elements with For Loop\*\*\*\*\*\*\*\*");

            // for loop to access list elements

            for (int i = 0; i < users.Count; i++)

            {

                Console.WriteLine("Id:{0}, Name:{1}, Location:{2}", users[i].Id, users[i].Name, users[i].Location);

            }

            Console.WriteLine("\*\*\*\*\*\*\*\*\*Access Elements with Foreach Loop\*\*\*\*\*\*\*\*");

            // foreach loop to access list elements

            foreach (User u in users)

            {

                Console.WriteLine("Id:{0}, Name:{1}, Location:{2}", u.Id, u.Name, u.Location);

            }

            Console.ReadLine();

        }

    }

}

If you observe above example, we are accessing generic list elements in different ways by using index positions, [for](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) and [foreach](https://www.tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "Foreach Loop in C# with Examples" \t "_blank) loops based on our requirements.

Try executing the program.

## C# Insert Elements into List

In c#, we can insert elements into the list either by using **Insert()** or **InsertRange()** methods.

Following is the example of inserting elements into the list by using **Insert** and **InsertRange** methods in c#.

using System;

using System.Collections.Generic;

namespace YourApp

{

    class Program

    {

        static void Main(string[] args)

        {

            // Creating and initializing list

            List<int> lst = new List<int>() { 1, 8, 45, 70 };

            // inserting elements into list

            lst.Insert(0, 10);

            lst.Insert(3, 50);

            List<int> lst2 = new List<int>() { 200, 300 };

            // inserting lst2 into lst at position 2

            lst.InsertRange(2, lst2);

            Console.WriteLine("\*\*\*\*\*\*\*\*\*ArrayList Elements\*\*\*\*\*\*\*\*");

            foreach (var item in lst)

            {

                Console.WriteLine(item);

            }

            Console.ReadLine();

        }

    }

}

If you observe above example, we inserted an elements into list at different index positions (**0**, **3**) by using **Insert()** method. Same way, by using **InsertRange()** method we are inserting all the element of newly created list (**lst2**) into **lst** at index position **2**.

Try executing the above program.

## C# Remove Elements from List

In c#, we have different methods to remove elements from the list i.e. either by using **Remove()** or **RemoveAt()** or **RemoveRange()** methods.

Following is the example of deleting the elements from the list in c# programming language.

using System;

using System.Collections.Generic;

namespace YourApp

{

    class Program

    {

        static void Main(string[] args)

        {

            // Creating and initializing list

            List<int> lst = new List<int>() { 10, 20, 30, 40, 50, 60, 70, 80 };

            // Removing an element which is having a value 50

            lst.Remove(50);

            // Removing an element at index 2

            lst.RemoveAt(2);

            // Removing 2 elements starting from index 3

            lst.RemoveRange(3, 2);

            Console.WriteLine("\*\*\*\*\*\*\*\*\*List Elements\*\*\*\*\*\*\*\*");

            foreach (var item in lst)

            {

                Console.WriteLine(item);

            }

            Console.ReadLine();

        }

    }

}

If you observe above example, we used a **Remove()** method to delete particular value of element from list and used a **RemoveAt()** method to delete an element at specified index position.

Same way, we used the **RemoveRange()** method to delete the specified number of elements from the list starting from the specified index position.

Try executing the above program.

In case, if you want to remove all the elements from the list, then use the **Clear()** method.

Same way, you can use different methods like **Find()** method to find elements of list and **Reverse()** method to reverse the order of list elements, etc. based on our requirements.

## C# List Check If Item Exists

 By using **Contains()** method, we can check whether the specified element exists in the list or not. In case, if it exists it will return **true** otherwise **false**.

Following is the example of using **Contains()** method to check for an item that exists in the list or not in c#.

using System;

using System.Collections.Generic;

namespace YourApp

{

    class Program

    {

        static void Main(string[] args)

        {

            // Creating and initializing list

            List<int> lst = new List<int>() { 10, 20, 30, 40, 50, 60, 70, 80 };

            // Check for an item 50 exists in list or not

            Console.WriteLine("Item Exists: " + lst.Contains(50));

            Console.ReadLine();

        }

    }

}

If you observe above example, we used a **Contains()** method to check for an item (**50**) exists in list (**lst**) or not.

When you execute the above c# program, we will get the result as shown below.

Item Exists: True

## C# List (List<T>) Overview

The following are the important points that need to remember about a list in c#.

* The list is used to store elements of the same [data type](https://www.tutlane.com/tutorial/csharp/csharp-data-types-with-examples) and the size of the list can grow or shrink dynamically by adding or removing elements.
* The list is a generic type of collection and available with the **System.Collections.Generic** [namespace](https://www.tutlane.com/tutorial/csharp/csharp-namespaces-with-examples).
* We can access the list elements either by using loops ([for](https://www.tutlane.com/tutorial/csharp/csharp-for-loop-with-examples) and [foreach](https://www.tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "C# Foreach Loop with Examples" \t "_blank)) or with a particular index position.
* **List<T>** can be assigned to **IList<T>** or **List<T>** type of a [variable](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) and you can store multiple null and duplicate values in the list based on your requirement.
* The list provided different methods to perform multiple operations like add, insert, delete, sort, reverse, etc. on elements of arraylist.

List vs Array: A Quick Guide That You Must Know About C#

In C#, arrays and lists are both objects that can be used to hold variables, but they aren’t interchangeable. Let’s explore list vs array.

An array must be declared using brackets and accompanied by the type of variables it will hold (integers or strings) and by its name. To declare an array of integers with the name numbers, it would need to look like this:

int[] numbers;

(you’d need to add a comma within the brackets if you’d like the array to be multi-dimensional).

To create the array, you’d have to add some code to instantiate it, like this:

int[] numbers = new int[];

## C# List

To create a list in C#, you need to call the list and put the type of list (again, integer or string) in angled brackets. Then follow that with the name of your list. That will declare a new C# list. Like an array, to create a list, you must instantiate it, as seen below:

List<string> Food = new List<string>();

In general, it’s better to use lists in C# because lists are far more easily sorted, searched through, and manipulated in C# than arrays. That’s because of all of the built-in list functionalities in the language.

Lists are used more often in C# than arrays are, but there are some instances where arrays can (or should) be used. That includes if your data is unlikely to grow very much or if you’re dealing with a relatively large amount of data that will need to be indexed into often.

## List vs Array: Final Thoughts

Let’s conclude the list vs array. Inserting parts in the middle of the list is exhausting since arrays are contiguous in memory. Lists allow straightforward insertion into lists. An array is a method of organizing data in a memory device. A list is a data structure that supports several operations. An array is a collection of homogenous parts, while a list consists of heterogeneous elements. Array memory is static and continuous. List memory is dynamic and random. Users don’t need to confine track of next memory with arrays. With lists, a user has to track of next location.

Hence, arrays and lists are useful in the c# programming language.