C# collection types are designed to store, manage and manipulate similar data more efficiently. Data manipulation includes adding, removing, finding, and inserting data in the collection. Collection types implement the following common functionality:

* Adding and inserting items to a collection
* Removing items from a collection
* Finding, sorting, searching items
* Replacing items
* Copy and clone collections and items
* Capacity and Count properties to find the capacity of the collection and number of items in the collection

.NET supports two types of collections, generic collections and non-generic collections. Prior to .NET 2.0, it was just collections and when generics were added to .NET, generics collections were added as well.

Generic collections with work generic data type. Learn more about generics here: [Generics in C#](https://www.c-sharpcorner.com/UploadFile/84c85b/using-generics-with-C-Sharp/).

The following table lists and matches these classes.

**Non-generic                          Generic**

 ArrayList     ------------->          List

 HashTable  ------------->          Dictionary

 SortedList   ------------->          SortedList

 Stack           ------------->          Stack

 Queue         ------------->          Queue

**What is ArrayList in C#?**

The **ArrayList in C#** is a collection class that works like an array but provides the facilities such as dynamic resizing, adding and deleting elements from the middle of a collection. It implements the System.Collections.IList interface using an array whose size is dynamically increased as required.

**Methods and Properties of ArrayList Collection class in C#:**

The following are the methods and properties provided by the ArrayList collection class in C#.

1. **Add(object value):**This method is used to add an object to the end of the collection.
2. **Remove(object obj):**This method is used to remove the first occurrence of a specific object from the collection.
3. **RemoveAt(int index):**This method takes the index position of the elements and remove that element from the collection.
4. **Insert(int index, Object value):**This method is used to inserts an element into the collection at the specified index.
5. **Capacity:** This property gives you the capacity of the collection means how many elements you can insert into the collection.

**Example of ArrayList Collection class in C#:**

Let us see an example by using the above methods and properties of the ArrayList class. The code is self-explained so please go through the comments.

**using** *System;*

**using** *System.Collections;*

**namespace** *ArrayListCollection*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Createing ArrayList collection using default constructor

ArrayList al = new ArrayList**()**;

Console.WriteLine**(**"Initial Capacity: " + al.Capacity**)**;

al.Add**(**10**)**;

Console.WriteLine**(**"Capacity after adding first item: " + al.Capacity**)**;

al.Add**(**"hello"**)**;

al.Add**(true)**;

al.Add**(**3.14f**)**;

Console.WriteLine**(**"Capacity after adding fourth item: " + al.Capacity**)**;

al.Add**(**'A'**)**;

Console.WriteLine**(**"Capacity after adding 5th element: " + al.Capacity**)**;

//Printing the ArrayList elements using for loop

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

**{**

Console.Write**(**al**[**i**]** + " "**)**;

**}**

Console.WriteLine**()**;

//Removing the values from the middle of the array list

//here we are removing by value

al.Remove**(true)**;

//You can also remove element by using index position

// al.RemoveAt(2);

//Printing the ArrayList elements using foreach loop after

// removing an element from the collection

**foreach** **(object** obj in al**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

Console.WriteLine**()**;

//inserting values into the middle of the array list collection

al.Insert**(**2, **false)**;

// Printing the values of the collection using foreach loop after

// inserting an element into the middle of the collection

**foreach** **(object** obj in al**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

Console.WriteLine**()**;

// creating new ArrayList collection by passing the old

// array list as parameter

ArrayList coll = new ArrayList**(**al**)**;

Console.WriteLine**(**"Initial capacity of new array list collection:" + coll.Capacity**)**;

// Printing the values of the new array list collection using foreach loop

**foreach** **(object** obj in coll**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

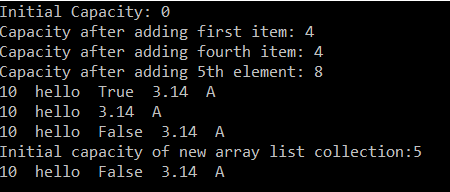
Console.ReadKey**()**;

**}**

**}**

**}**

**OUTPUT:**



**What is the difference between an array and an Array List in C#?**

This is one of the frequently asked interview questions in C#. So let us discuss the difference between an array and ArrayList.

**Array:**

1. Fixed Length
2. Cannot insert into the middle
3. Cannot delete from middle

**ArrayList:**

1. Variable Length
2. Can insert into the middle of the collection
3. Can remove into the middle of the collection

## **Hashtable in C# with Examples**

##### ****Problems with Array and ArrayList Collection in C#:****

In the case of [**Array**](https://dotnettutorials.net/lesson/arrays-csharp/)and **[ArrayList](https://dotnettutorials.net/lesson/arraylist-collection-csharp/)**in C#, we can access the elements from the collection using a key. That key is nothing but the index position of the elements which is starts from zero (0) to the number of elements – 1. But in reality, it’s very difficult to remember the index position of the element in order to access the values.

For example, let’s say we have an employee array that contains the name, address, mobile, dept no, email id, employeeid, salary, location, etc. Now if I want to know the email id or dept no of the employee then it’s very difficult for me. This is because we need to access the value by using the index position. The following example shows this. here we create a collection using ArrayList and then we are accessing the Location and EmailId by using the index position.

**namespace** *HasntableExample*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

ArrayList al = new ArrayList**()**;

al.Add**(**1001**)**; //EId

al.Add**(**"James"**)**; //Name

al.Add**(**"Manager"**)**; //Job

al.Add**(**3500**)**; //Salary

al.Add**(**"Mumbai"**)**; //Location

al.Add**(**"IT"**)**; //Dept

al.Add**(**"a@a.com"**)**; // Emailid

//I want to print the Location, index position is 4

Console.WriteLine**(**"Location : " + al**[**4**])**;

//I want to print the Email ID, index position is 6

Console.WriteLine**(**"Emaild ID : " + al**[**6**])**;

Console.ReadKey**()**;

**}**

**}**

**}**

You have a huge number of elements in the collection, then it is very difficult to remember the index position of each element. So, it would not be nice, instead of using the index position of the element, if we can access the elements by using a key. This is where Hashtable in C# comes into the picture.

##### ****What is a Hashtable in C#?****

The **Hashtable in C#** is a collection that stores the element in the form of “**Key-Value pairs**”. The data in the Hashtable are organized based on the hash code of the key. The key in the HashTable is defined by us and more importantly, that key can be of any data type. Once we created the Hashtable collection, then we can access the elements by using the keys. The Hashtable class comes under the **System.Collections** namespace.

Note: The Hashtable computes a hash code for each key. Then it uses that hash code to lookup for the elements very quickly which increases the performance of the application.

**Let us see an example for a better understanding of the Hashtable in C#.**

**using** *System;*

**using** *System.Collections;*

**namespace** *HasntableExample*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Creating Hashtable collection with default constructor

Hashtable ht = new Hashtable**()**;

//Adding elements to the Hash table using key value pair

ht.Add**(**"EId", 1001**)**; //Here key is Eid and value is 1001

ht.Add**(**"Name", "James"**)**; //Here key is Name and value is James

ht.Add**(**"Job", "Developer"**)**;

ht.Add**(**"Salary", 3500**)**;

ht.Add**(**"Location", "Mumbai"**)**;

ht.Add**(**"Dept", "IT"**)**;

ht.Add**(**"EmailID", "a@a.com"**)**;

//Printing the keys and their values using foreach loop

Console.WriteLine**(**"Printing using Foreach loop"**)**;

**foreach** **(object** obj in ht.Keys**)**

**{**

Console.WriteLine**(**obj + " : " + ht**[**obj**])**;

**}**

Console.WriteLine**()**;

Console.WriteLine**(**"Printing using Keys"**)**;

//I want to print the Location by using the Location key

Console.WriteLine**(**"Location : " + ht**[**"Location"**])**;

//I want to print the Email ID by using the EmailID key

Console.WriteLine**(**"Emaild ID : " + ht**[**"EmailID"**])**;

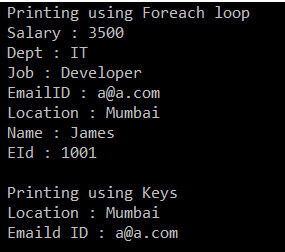
Console.ReadKey**()**;

**}**

**}**

**}**

###### **OUTPUT:**



When you are working with the Hashtable, then you need to understand two important methods are as follows.

###### **ContainsKey() Method:**

The ContainsKey() method of the Hashtable is used to check if a given key is present in the Hashtable or not. The following is the syntax for using the ContainsKey() method. If the given key is present in the collection then it will return true else it will return false.

**Hashtable.Containskey(key)**

###### **ContainsValue() Method:**

The ContainsValue() Method of the Hashtable class is used to check if a value is present in the Hashtable or not. The following is the syntax for using the ContainsValue() Method. If the given value is present in the collection then it will return true else it will return false.

**Hashtable.ContainsValue(value)**

**Let us see an example for a better understanding of the above two methods:**

**using** *System;*

**using** *System.Collections;*

**namespace** *HasntableExample*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Creating Hashtable collection with default constructor

Hashtable ht = new Hashtable**()**;

//Adding elements to the Hash table using key value pair

ht.Add**(**"EId", 1001**)**; //Here key is Eid and value is 1001

ht.Add**(**"Name", "James"**)**; //Here key is Name and value is James

ht.Add**(**"Job", "Developer"**)**;

ht.Add**(**"Salary", 3500**)**;

ht.Add**(**"Location", "Mumbai"**)**;

ht.Add**(**"Dept", "IT"**)**;

ht.Add**(**"EmailID", "a@a.com"**)**;

//Checking the key using the ContainsKey methid

Console.WriteLine**(**"Is EmailID Key Exists : " + ht.ContainsKey**(**"EmailID"**))**;

Console.WriteLine**(**"Is Department Key Exists : " + ht.ContainsKey**(**"Department"**))**;

//Checking the value using the ContainsValue method

Console.WriteLine**(**"Is Mumbai value Exists : " + ht.ContainsValue**(**"Mumbai"**))**;

Console.WriteLine**(**"Is Technology value Exists : " + ht.ContainsValue**(**"Technology"**))**;

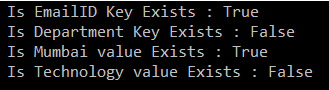
Console.ReadKey**()**;

**}**

**}**

**}**

**Output:**



## **Stack in C# with Examples**

##### ****What is Stack in C#?****

The **Stack in C#** is a non-generic collection class that works in the **LIFO (Last In First Out)** principle. So, we need to use the Stack Collection in C#, when we need last in first out access to the items of a collection. That means the item which is added last will be removed first. When we add an item into the stack, then it is called as pushing an item. Similarly when we remove an item from the stack then it is called popping an item. The Stack class belongs to the **System.Collections** namespace.

Let us understand the LIFO principle with an example. Imagine we have a stack of plates with each plate added on top of each other. The last plate which is added on the stack will be the first one to remove from the stack. It is not possible to remove a plate from the middle of the stack.

**Note:** The **Stack Collection in C#** allows both null and duplicate values.

##### ****Methods of Stack class in C#:****

**Push():**The push() method is used to Inserts an object on top of the Stack.

**Syntax: void Stack.Push(Object obj)**

**Pop():**The **pop()** method is used to remove and return the object at the top of the Stack. If there is no object (or element) present in the stack and if you are trying to remove an item or object from the stack using the pop() method then it will throw an exception i.e. **System.InvalidOperationException**

**Syntax: Object stack.pop()**

**Peek():**The peek() method is used to return the object from the top of the Stack without removing it. If there is no object (or element) present in the stack and if you are trying to return an item (object) from the stack using the peek() method then it will throw an exception i.e. **System.InvalidOperationException**

**Syntax: object Stack.Peek()**

###### **Example: Let us understand the above methods of Stack class using an example.**

**using** *System;*

**using** *System.Collections;*

**namespace** *StackCollectionDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Creating a stack collection

Stack s = new Stack**()**;

//Adding item to the stack using the push method

s.Push**(**10**)**;

s.Push**(**"hello"**)**;

s.Push**(**3.14f**)**;

s.Push**(true)**;

s.Push**(**67.8**)**;

s.Push**(**'A'**)**;

//Printing the stack items using foreach loop

**foreach** **(object** obj in s**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

Console.WriteLine**()**;

//Removing annd returning an item from the stack

//using the pop method

Console.WriteLine**(**s.Pop**())**;

Console.WriteLine**()**;

//Printing item after removing the last added item

**foreach** **(object** obj in s**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

Console.WriteLine**()**;

//Returning the last item from the stack without removing it

//by using the peek method

Console.WriteLine**(**s.Peek**())**;

Console.WriteLine**()**;

//Printing the items after using the Peek method

**foreach** **(object** obj in s**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

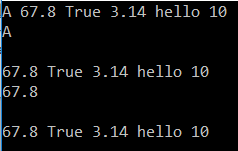
Console.ReadKey**()**;

**}**

**}**

**}**

###### **OUTPUT:**



##### ****Let us discuss some other methods and properties of Stack Class:****

**Count:** The Count property of the Stack class is used to return the number of elements present in the Stack.  
**Syntax:** **Stack.Count**

**Contains():** The Contains() method of the Stack class is used to check whether an element is present in the Stack or not. If it presents, then it will return true else it will return false.  
**Syntax:** **Stack.Contains(element)**

**Clear():** The Clear() method of the Stack class is used to remove all the elements from the stack.  
**Syntax:** **Stack.Clear()**

###### **Example: Let us see an example for a better understanding of the above methods and properties**

**using** *System;*

**using** *System.Collections;*

**namespace** *StackCollectionDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Creating a stack collection

Stack s = new Stack**()**;

//Adding item to the stack using the push method

s.Push**(**20**)**;

s.Push**(**"hi"**)**;

s.Push**(**3.14f**)**;

s.Push**(true)**;

s.Push**(**12.3**)**;

s.Push**(**'P'**)**;

//Printing the stack items using foreach loop

**foreach** **(object** obj in s**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

Console.WriteLine**()**;

//Using Count property to get the number of items

//present in the collection

Console.WriteLine**(**$"No of Elements in the Collection : {s.Count}"**)**;

Console.WriteLine**()**;

//Using the Contains method to check whether an item is present or not

Console.WriteLine**(**$"Is the value hi present in the collection : {s.Contains("hi")}"**)**;

Console.WriteLine**()**;

//Removing all the items from the collection using Clear() method

s.Clear**()**;

Console.WriteLine**(**$"No of Elements in the Collection after Clear() method : {s.Count}"**)**;

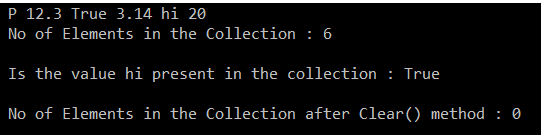
Console.ReadKey**()**;

**}**

**}**

**}**

**Output:**



## **Queue in C# with Examples**

##### ****What is a Queue in C#?****

The Queue in C# is a non-generic collection class that works in the **FIFO (First In First Out)** principle. So, we need to use the Queue Collection in C#, when we need the first in first out access to the items of a collection. That means the item which is added first will be removed first from the collection. When we add an item into the queue collection, it is called as enqueuing an item. Similarly when we remove an item from the queue collection then it is called dequeuing an item. The Queue class belongs to the System.Collections namespace.

Let us understand the FIFO principle with an example. Imagine a queue of people waiting for the ticket in a cinema hall. Normally, the first person who enters the queue will be the first person to get the ticket from the counter. Similarly, the last person who enters into the queue will be the last person to get the ticket from the counter.

**Note:** The non-generic **Queue Collection class in C#** allows both null and duplicate values.

##### ****Methods of Queue class in C#:****

**Enqueue():** This method is used to add an item (or object) to the end of the Queue.  
**Syntax:** **void Queue.Enqueue(object obj)**

**Dequeue():** The Dequeue() method of the Queue class is used to Remove and return the object from the beginning of the Queue. If there is no object (or element) present in the Queue and if we are trying to remove an item or object from the Queue using the pop() method then it will throw an exception i.e. **System.InvalidOperationException**  
**Syntax:** **object Queue.Dequeue()**

**Peek():** The peek() method of the Queue class is used to return the oldest object i.e. the object present at the start of the Queue without removing it. If there is no object (or element) present in the Queue and if we are trying to return an item (object) from the Queue using the peek() method then it will throw an exception i.e. **System.InvalidOperationException**

**Syntax:** **object Queue.Peek()**

##### ****Example: Let us understand the above methods of Queue class with an example.****

**using** *System;*

**using** *System.Collections;*

**namespace** *QueueCollectionDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Creating a queue collection

Queue q = new Queue**()**;

//Adding item to the queue using the Enqueue method

q.Enqueue**(**10**)**;

q.Enqueue**(**"hello"**)**;

q.Enqueue**(**3.14f**)**;

q.Enqueue**(true)**;

q.Enqueue**(**67.8**)**;

q.Enqueue**(**'A'**)**;

//Printing the queue items using foreach loop

**foreach** **(object** obj in q**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

Console.WriteLine**()**;

//Removing and returning an item from the queue

//using the Dequeue method

Console.WriteLine**(**q.Dequeue**())**;

Console.WriteLine**()**;

//Printing item after removing the first added item

**foreach** **(object** obj in q**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

Console.WriteLine**()**;

//Returning the first item from the queue without removing it

//by using the peek method

Console.WriteLine**(**q.Peek**())**;

Console.WriteLine**()**;

//Printing the items after using the Peek method

**foreach** **(object** obj in q**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

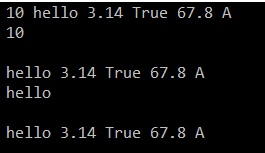
Console.ReadKey**()**;

**}**

**}**

**}**

**OUTPUT:**



##### ****Let us discuss some other important methods and properties of Queue Class:****

**Count:** The Count property of the Queue class is used to return the number of elements present in the Queue Collection.  
**Syntax:** **Queue.Count**

**Contains():** The Contains() method of the Queue class is used to check whether an object (element) is present in the Queue or not. If it presents, then it will return true else it will return false.  
**Syntax:** **Queue.Contains(element)**

**Clear():** The Clear() method of the Queue class is used to remove all the elements from the queue collection.  
**Syntax: Queue.Clear()**

###### **Example: Let us see an example for a better understanding of the above methods and properties of Queue Class**

**using** *System;*

**using** *System.Collections;*

**namespace** *QueueCollectionDemo*

**{**

**class** Program

**{**

**static** **void** Main**(**string**[]** args**)**

**{**

//Creating a queue collection

Queue q = new Queue**()**;

//Adding item to the qieue using the Enqueue method

q.Enqueue**(**10**)**;

q.Enqueue**(**"hi"**)**;

q.Enqueue**(**3.14f**)**;

q.Enqueue**(true)**;

q.Enqueue**(**67.8**)**;

q.Enqueue**(**'A'**)**;

//Printing the queue items using foreach loop

**foreach** **(object** obj in q**)**

**{**

Console.Write**(**obj + " "**)**;

**}**

Console.WriteLine**()**;

//Using Count property to get the number of items

//present in the queue collection

Console.WriteLine**(**$"No of Elements Present in the Collection : {q.Count}"**)**;

Console.WriteLine**()**;

//Using the Contains method to check whether an item is present or not

Console.WriteLine**(**$"Is the value hi present in the collection : {q.Contains("hi")}"**)**;

Console.WriteLine**()**;

//Removing all the items from the collection using Clear() method

q.Clear**()**;

Console.WriteLine**(**$"No of Elements in the Collection after Clear() method : {q.Count}"**)**;

Console.ReadKey**()**;

**}**

**}**

**}**

**Output:**

