Collections in C#

**Introduction to Collections in C#**

For many applications, you want to create and manage groups of related objects. There are two ways to group objects:

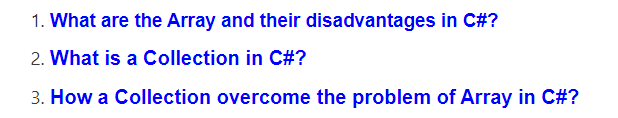
by creating arrays of objects,

and by creating collections of objects.

Arrays are most useful for creating and working with a fixed number of strongly typed objects. For information about arrays, see [Arrays](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/).

Collections provide a more flexible way to work with groups of objects. Unlike arrays, the group of objects you work with can grow and shrink dynamically as the needs of the application change. For some collections, you can assign a key to any object that you put into the collection so that you can quickly retrieve the object by using the key.

A collection is a class, so you must declare an instance of the class before you can add elements to that collection.



So, let’s first understand what is the problem with the traditional array in C#, and then we will discuss how to overcome the problems using collections in C#.

**What are the Array and their disadvantages in C#?**

In simple words, we can say that the Arrays in C# are the simple data structure that is used to store similar types of data items in sequential order.

Although the arrays in c# are commonly used, they have some limitations.

For example, you need to specify the array’s size while creating the array.

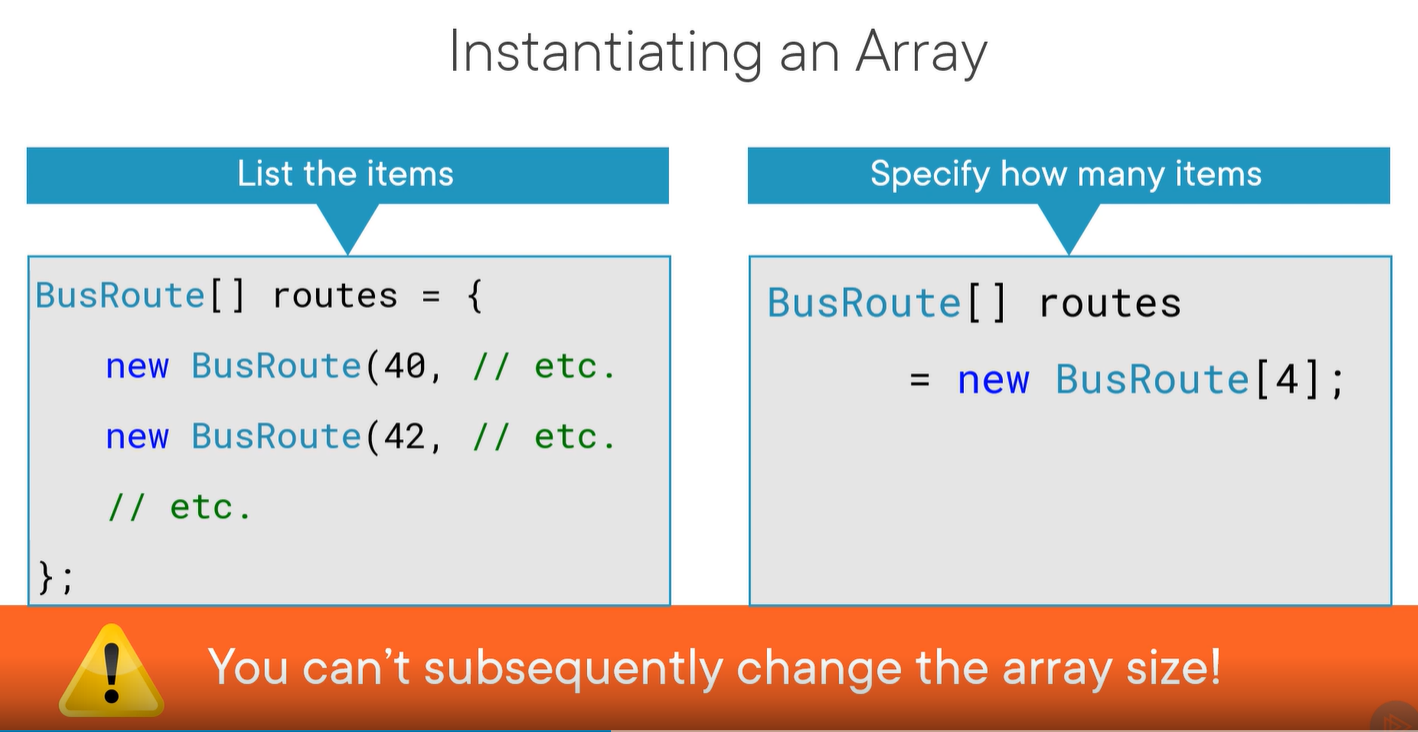
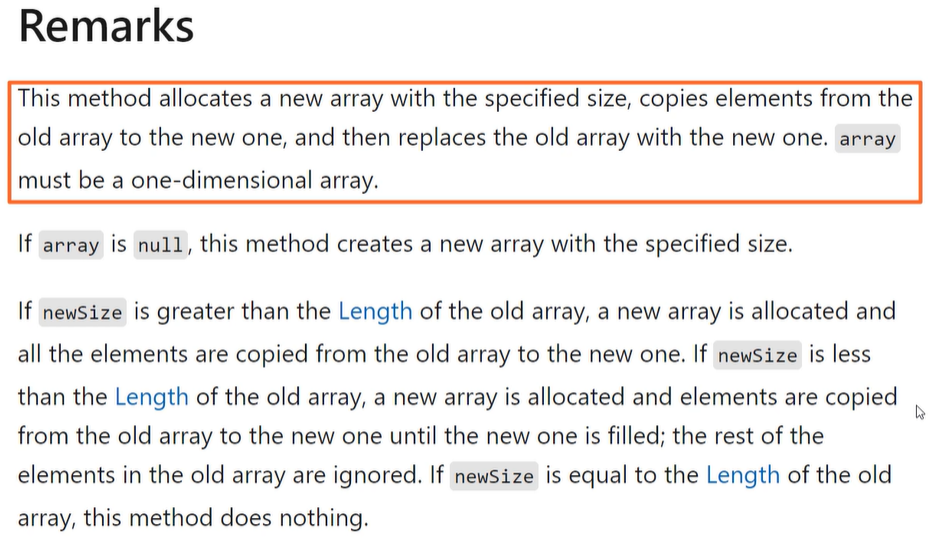
* Array cant grow its size automatically
* Int[] numbers=new int[4]

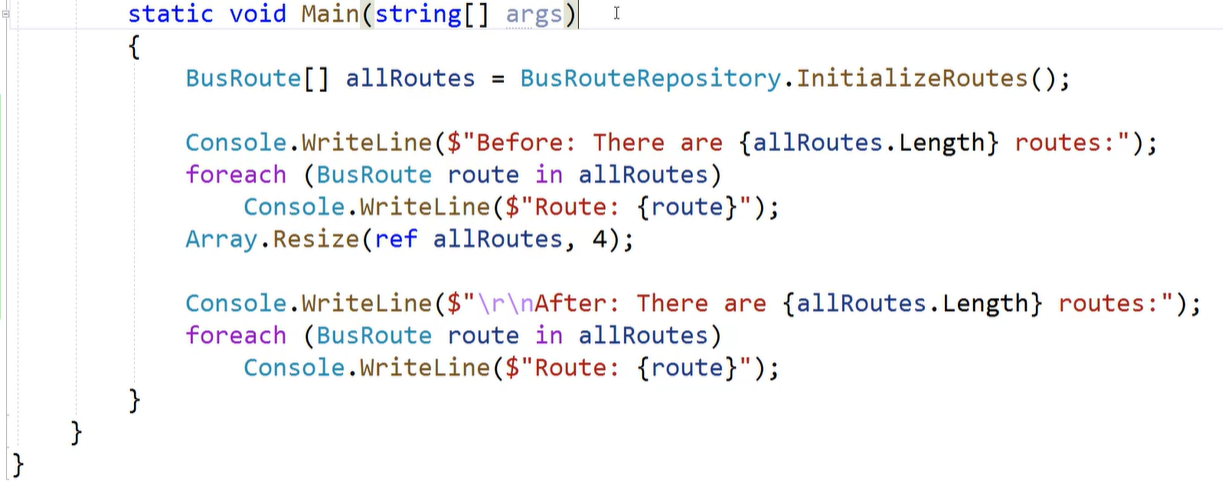
If at execution time, you want to modify it that means if you want to increase or decrease the size of an array, then you need to do it manually by creating a new array or by using the Array class’s Resize method, which internally creates a new array and copies the existing array element into the new array.

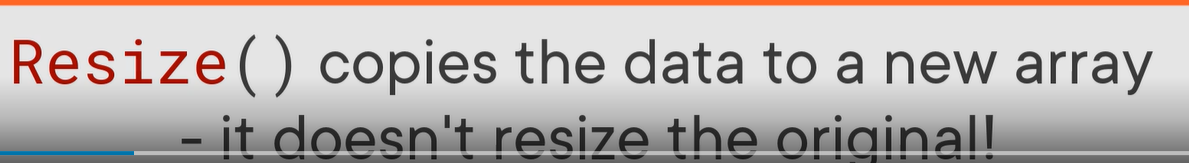
**Following are the limitations of Array in C#:**

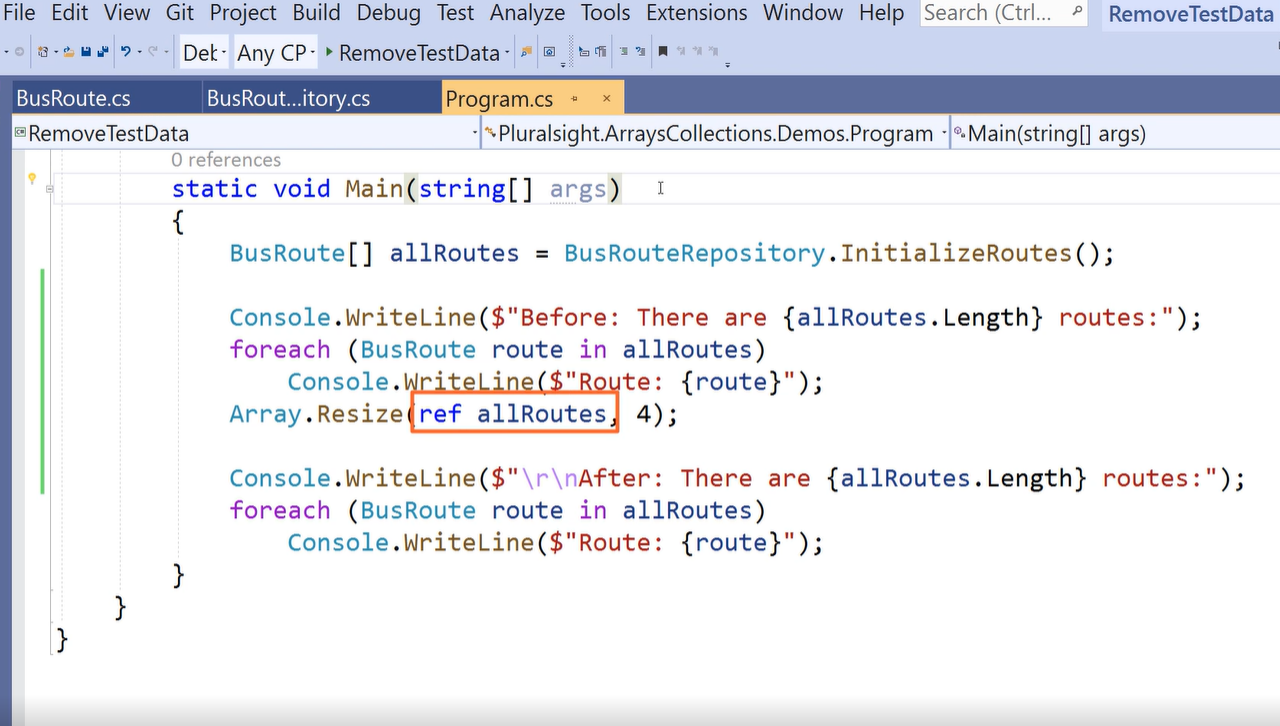
1. The array size is fixed. Once the array is created we can never increase the size of an array. If we want then we can do it manually by creating a new array and copy the old array elements into the new array or by using the Array class Resize method which will do the same thing means to create a new array and copy the old array elements into the new array and then destroy the old array.
2. We can never insert an element into the middle of an array
3. Deleting or removing elements from the middle of the array.

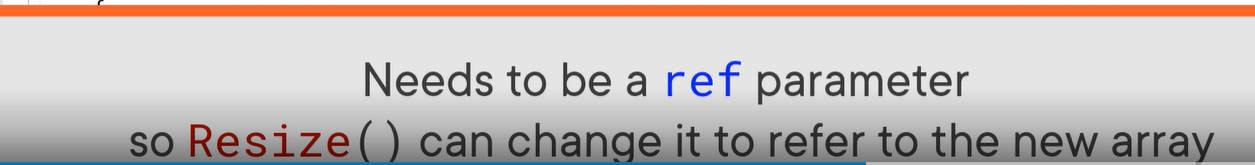
To overcome the above problems, the Collections are introduced in C# 1.0.

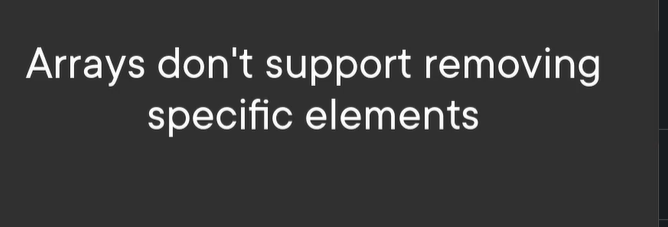
* We can use resize to resize the array with resize method but when you resize it
* It creates a new copy of memory and copies old values to new memory
* 









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

The **Collections in C#** are a set of predefined classes that are present in the **System.Collections** namespace that provides greater capabilities than the traditional arrays

. The collections in C# are reusable, more powerful, more efficient and most importantly they have been designed and tested to ensure quality and performance.

So in simple words, we can say a **Collection in C#** is a **dynamic array.** That means the collections in C# have the capability of storing multiple values but with the following features.

1. Size can be increased dynamically.
2. We can insert an element into the middle of a collection.
3. It also provides the facility to remove or delete elements from the middle of a collection.

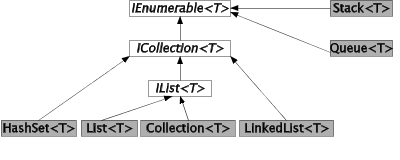
The collection which is from .Net framework 1.0 is called simply **collections** or **Non-Generic collections in C#**. These collection classes are present inside the **System.Collections** namespace. The example includes are **Stack, Queue, LinkedList, SortedList, ArrayList, HashTable,** etc.

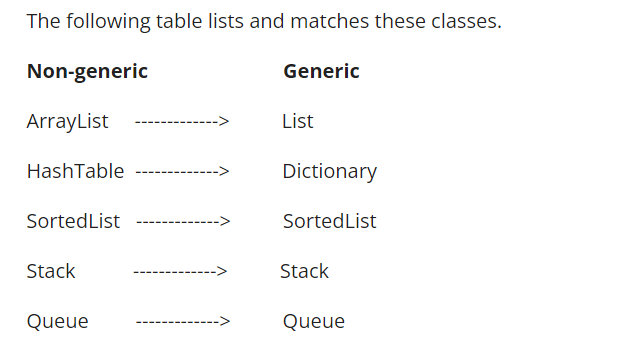
**Auto-Resizing of collections:**

The capacity of a collection increases dynamically i.e. when we keep adding new elements, then the size of the collection keeps increasing automatically. Every collection class has three constructors and the behavior of collections will be as following when created using a different constructor.

1. **Default Constructor:** It Initializes a new instance of the collection class that is empty and has the default initial capacity as zero which becomes four after adding the first element and whenever needed the current capacity becomes double.
2. **Collection (int capacity):** This constructor initializes a new instance of the collection class that is empty and has the specified initial capacity, here also when the requirement comes current capacity doubles.
3. **Collection (Collection):** It Initializes a new instance of the collection class that contains elements copied from the specified collection and that has the same initial capacity as the number of elements copied, here also when the requirement comes current capacity doubles.

C# Collections





In C#, collection represents group of objects. By the help of collections, we can perform various operations on objects such as

* store object
* update object
* delete object
* retrieve object
* search object, and
* sort objec
* **System.Collections.Generic** classes
* **System.Collections** classes (Now deprecated)
* **System.Collections.Concurrent** classes

### 1) System.Collections.Generic classes

The System.Collections.Generic namespace has following classes:

* List
* Stack
* Queue
* LinkedList
* HashSet
* SortedSet
* Dictionary
* SortedDictionary
* SortedList

### 2) System.Collections classes

These classes are legacy. It is suggested now to use System.Collections.Generic classes. The System.Collections namespace has following classes:

* ArrayList
* Stack
* Queue
* Hashtable

### 3) System.Collections.Concurrent classes

The System.Collections.Concurrent namespace provides classes for thread-safe operations. Now multiple threads will not create problem for accessing the collection items.

The System.Collections.Concurrent namespace has following classes:

* BlockingCollection
* ConcurrentBag
* ConcurrentStack
* ConcurrentQueue
* ConcurrentDictionary
* Partitioner
* Partitioner

# oxing and Unboxing in C#.Net

Before going through **Boxing**and **Unboxing** concept in C#, it’s important to know about value type and reference type variables. Please go through my article [here](https://thedotnetguide.com/csharp-boxing-and-unboxing/).

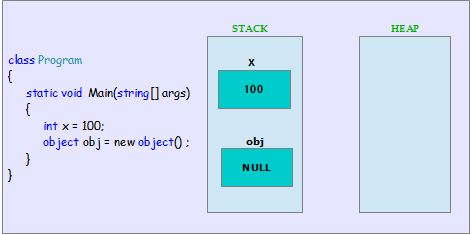
## ****Boxing**** :

It is the process of converting a **value**type variable to a **reference**type variable (object). That means the C#.Net converts a value type variable of type (int, float, bool etc) to an object which is a  reference type variable.

In below code snippet, we have declared a value type int x and then performing a boxing operation that converts the value type   x  to an object  obj  which is a reference type.

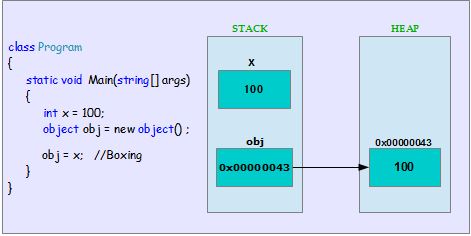
|  |  |
| --- | --- |
| 1  2  3 | int x = 100; //Value type  object obj = new object(); //Reference type  obj = x; //Boxing |

If we represent graphically, below image explains the memory allocation process for above code snippet. If you look closely,  we have declared a value type int x = 100 that is stored in the stack memory and the reference type obj variable is stored in the stack. (value of reference type stores in heap memory.)

Figure – 1 – Boxing

### ****How the framework handle memory allocation during Boxing ?****

During Boxing operation, when the compiler hits line obj = x  Here it determines that a **Boxing** operation has to be performed. Since we are trying to assign a **value** type to a **reference** type variable.

Figure – 2 – Boxing Operation

The compiler copies the integer value (**100**) from the stack, wraps up it as an object and allocates memory from the managed heap. It stores the memory address (memory address i.e **0x00000043**) in the stack.

If you look at closely, now the reference type variable obj is pointing to the integer variable which is stored in the managed heap (memory address i.e  **0x00000043**).

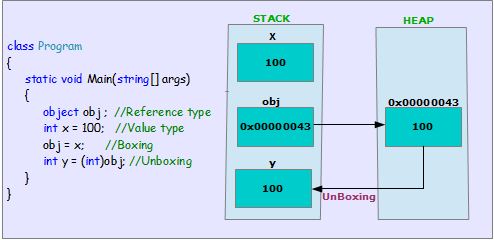
So internally, during Boxing operation, the compiler allocates memory space from managed heap for the value type’s field. It then copies the value in to the new memory location. Value type is stored as a Reference type object in new memory location. Now the memory address is copied to the stack, which refers to the object.

## ****Unboxing :****

It is the process of converting a **reference** type (Object) variable in to a **value** type variable. That means the C#.Net unwraps the reference type variable (object) to a value type variable(int, float, bool etc ).

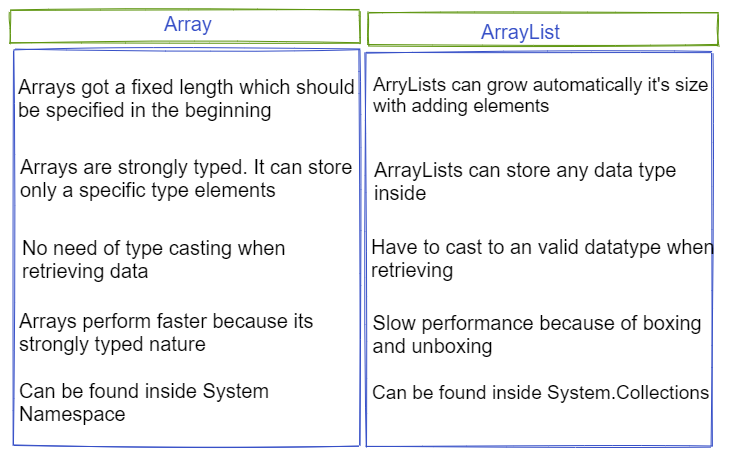
|  |  |
| --- | --- |
| 1  2  3  4 | object obj ; //Reference type  int x = 100; //Value type  obj = x; //Boxing  int y = (int)obj; //Unboxing |

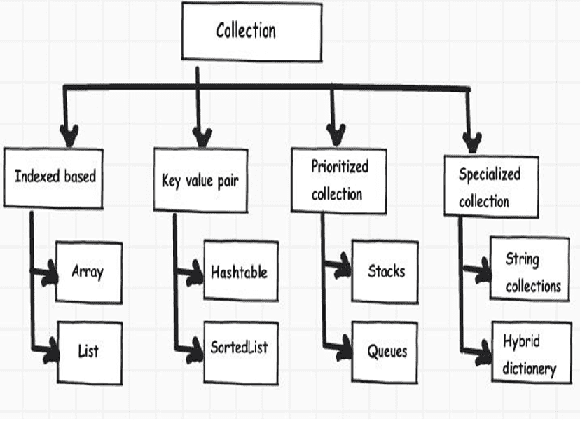
Extending the Boxing code snippet , In this above example, we are performing a Boxing operation and then, we have declared a value type variable int y  and assigning a reference type variable obj to it by doing the type cast.  The variable obj points to the boxed integer object int x in managed heap.

Figure – 3 – Unboxing

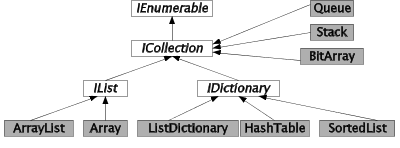
When the compiler hits the line int y = (int)obj .  It determines that it has to perform an **Unboxing**operation on the reference type (**obj**).  The compiler copies the  reference type’s  (**obj**) actual value (**100)** from object’s managed heap memory ( i.e – **0x00000043** ) and stores in the stack memory for the variable  y  which is a value type.

So internally, during the Unboxing operation, the compiler takes the boxed value type object which is on the managed heap memory. It  unwraps the value type’s value and copies it back to the stack memor

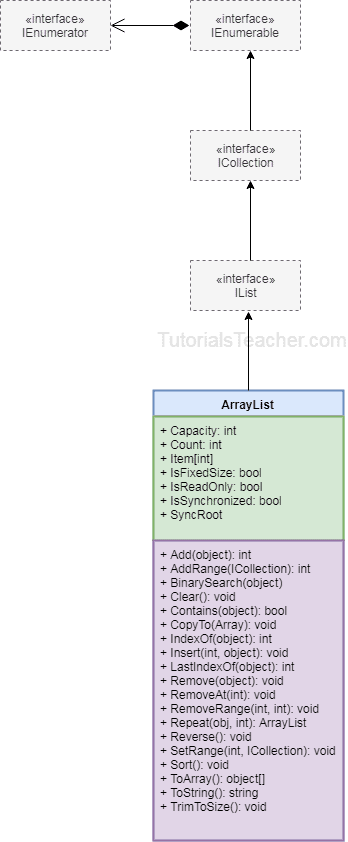




The non-generic collection classes store data of type **Object**



# ArrayList



ArrayList is a part of collections in C#. It is used to contain data of any given data type. It is similar to an array in C# but it doesn’t have a definite size. Its size increases automatically as more elements are added in it.

How To Initialize An ArrayList?

ArrayList can be initialized by using the “ArrayList” keyword.

ArrayList arrList = new ArrayList();

How To Add Elements Into An ArrayList?

You can add a single item into an array list by using the Add() method and a range of elements or multiple elements from another collection by using the AddRange() method.

Example:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Collections;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

ArrayList arrList = new ArrayList();

arrList.Add(7896);

arrList.Add("Seven");

Console.ReadLine();

}

}

}

Here, you can see that we have used ArrayList.Add() method. As you can notice, we have added both integers as well as string in the same array list. This is possible as an array list is of non-generic data type, i.e. it can contain an element of any given data type.

How To Access An Element From An ArrayList?

An array list element can be accessed similar to an array i.e. by using the index. But as it’s a non-generic type, we first need to cast it to a suitable data type or need to use var keyword while accessing a value from the Array List.

Example:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Collections;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

ArrayList arrList = new ArrayList();

arrList.Add(7896);

arrList.Add("Seven");

//casted and retrieved data

int first\_index = (int)arrList[0];

string second\_index = (string)arrList[1];

Console.WriteLine("The first index value is : " + first\_index);

Console.WriteLine("The second index value is : " + second\_index);

Console.ReadLine();

}

}

}

In the above example, we have retrieved the data by using the index of the array list and then we have cast it to the appropriate data type. The results have then been printed to the console as output.

The output for the above program will be:

Output

The first index value is: 7896

The second index value is: Seven

How To Insert An Element Into ArrayList?

To insert an element in the ArrayList at a specified point or index. the Insert() method is used.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Collections;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

ArrayList arrList = new ArrayList();

arrList.Add(7896);

arrList.Add("Seven");

//casted and retrieved data

int first\_index = (int)arrList[0];

string second\_index = (string)arrList[1];

Console.WriteLine("The first index value is : " + first\_index);

Console.WriteLine("The second index value is : " + second\_index);

arrList.Insert(1, "Eight");

second\_index = (string)arrList[1];

Console.WriteLine("The second index value is : " + second\_index);

Console.ReadLine();

}

}

}

Thus, we have inserted a new string at index 1 using the insert() method. So, if we run the above program we will get the following output:

Output

The first index value is: 7896

The second index value is: Seven

The second index value is: Eight

How To Remove An Element From ArrayList?

An element can be removed from the ArrayList using the Remove() method. Remove method accepts a parameter i.e. the value that needs to be removed from the array.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Collections;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

ArrayList arrList = new ArrayList();

arrList.Add(7896);

arrList.Add("Seven");

//casted and retrieved data

int first\_index = (int)arrList[0];

string second\_index = (string)arrList[1];

Console.WriteLine("The first index value is : " + first\_index);

Console.WriteLine("The second index value is : " + second\_index);

arrList.Insert(1, "Eight");

second\_index = (string)arrList[1];

Console.WriteLine("The second index value is : " + second\_index);

arrList.Remove(7896);

var data = arrList[0];

Console.WriteLine("The value at 0 index is : " + data);

Console.ReadLine();

}

}

}

The remove method removes the given value from the list. When the value is removed from the given index, the subsequent value moves one index upward to fill the gap. As we are removing 0 indexes, the value from index 1 will move and fill the void at 0.

The output of the following program will be:

Output

The first index value is: 7896

The second index value is: Seven

The second index value is: Eight

The value at 0 index is: Eight

How To Remove List Element Using The Index?

The user can remove a list element from a particular index by using the RemoveAt() method. RemoveAt() accepts a single parameter with an index number from which the element has to be removed. Similar to the Remove method, removing an element from the middle will push the next element to move one step upward to fill the gap.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Collections;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

ArrayList arrList = new ArrayList();

arrList.Add(7896);

arrList.Add("Seven");

//casted and retrieved data

int first\_index = (int)arrList[0];

string second\_index = (string)arrList[1];

Console.WriteLine("The first index value is : " + first\_index);

Console.WriteLine("The second index value is : " + second\_index);

arrList.Insert(1, "Eight");

second\_index = (string)arrList[1];

Console.WriteLine("The second index value is : " + second\_index);

arrList.RemoveAt(1);

var data = arrList[1];

Console.WriteLine("The value at 1 index is : " + data);

Console.ReadLine();

}

}

}

In the above example, we have used, RemoveAt to remove index 1. Hence, the element at index 2 should move to index 1 to replace the gap.

The output of the following program will be:

Output

The first index value is : 7896

The second index value is : Seven

The second index value is : Eight

The value at 1 index is : Seven

How To Sort And Reverse An ArrayList?

ArrayList offers two different methods for sorting and reverse operations. There is only one condition i.e. all the elements inside the array list should have the same data type for comparison with a comparator or else it will throw a runtime error.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Collections;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

ArrayList arrList = new ArrayList();

arrList.Add(7);

arrList.Add(4);

arrList.Add(5);

arrList.Add(1);

arrList.Add(3);

Console.WriteLine("Original Array List");

foreach (var v in arrList)

{

Console.Write(v + " ");

}

//sorting an array list

Console.WriteLine();

Console.WriteLine("Sorted Array List");

arrList.Sort();

foreach (var srt in arrList)

{

Console.Write(srt + " ");

}

//Reversing an array list

Console.WriteLine();

Console.WriteLine("Reversed Array List");

arrList.Reverse();

foreach (var rvrs in arrList)

{

Console.Write(rvrs + " ");

}

Console.ReadLine();

}

}

}

In the above program, we have first created an array list with integer data type and then we have used the sort method to sort the list and the reverse method to reverse the sorted list.

So, the output of the following list will be:

Output

Original Array List

7 4 5 1 3

Sorted Array List

1 3 4 5 7

Reversed Array List

# # ArrayList BinarySearch(Object, IComparer)

### Description

ArrayList BinarySearch(Object, IComparer) searches the entire sorted ArrayList for an element using the specified comparer and returns the zero-based index of the element.

### Syntax

ArrayList.BinarySearch(Object, IComparer) has the following syntax.

**public** virtual **int** BinarySearch(

Object value,

IComparer comparer

)

### Parameters

ArrayList.BinarySearch(Object, IComparer) has the following parameters.

* value - The Object to locate. The value can be null.
* comparer - The IComparer implementation to use when comparing elements.
* comparer - -or-
* comparer - null to use the default comparer that is the IComparable implementation of each element.

### Returns

ArrayList.BinarySearch(Object, IComparer) method returns The zero-based index of value in the sorted ArrayList, if value is found; otherwise, a negative number, which is the bitwise complement of the index of the next element that is larger than value or, if there is no larger element, the bitwise complement of Count.

### Example

The following example creates an ArrayList of colored animals. The provided IComparer performs the string comparison for the binary search.

/\* www .j a v a2 s.com\*/

**using** System;

**using** System.Collections;

**public** **class** SimpleStringComparer : IComparer

{

**int** IComparer.Compare(**object** x, **object** y)

{

**string** cmpstr = (**string**)x;

**return** cmpstr.CompareTo((**string**)y);

}

}

**public** **class** MyArrayList : ArrayList

{

**public** **static** **void** Main()

{

MyArrayList coloredAnimals = **new** MyArrayList();

coloredAnimals.Add(**"W"**);

coloredAnimals.Add(**"P"**);

coloredAnimals.Add(**"R"**);

coloredAnimals.Add(**"G"**);

coloredAnimals.Add(**"B"**);

coloredAnimals.Add(**"A"**);

coloredAnimals.Add(**"L"**);

coloredAnimals.Sort();

**int** index = coloredAnimals.BinarySearch(**"W"**, **new** SimpleStringComparer());

Console.WriteLine(**"Binary search, item found at index: {0}"**, index);

}

}

The code above generates the following result.

**C# HashTable**

* [ADING](http://dotnetpattern.com/threading-synchronization)
* [WPF](http://dotnetpattern.com/category/wpf)
* [DESIGN PATTERNS](http://dotnetpattern.com/introduction-to-design-patterns)
* [LINQ](http://dotnetpattern.com/LINQ-tutorials)
* [ANGULARJS](http://dotnetpattern.com/introduction-angularjs)
* [TYPESCRIPT](http://dotnetpattern.com/typescript-tutorial)
* [TESTING](http://dotnetpattern.com/category/Unit-Testing-Frameworks)
* [INTERVIEW QUESTIONS](http://dotnetpattern.com/category/interview-questions)

##### Tags

[**WPF**](http://dotnetpattern.com/tag/wpf) [**C#**](http://dotnetpattern.com/tag/CSharp) [**INTERVIEW**](http://dotnetpattern.com/tag/interview) [**REFLECTION**](http://dotnetpattern.com/tag/reflection) [**MVVM**](http://dotnetpattern.com/tag/mvvm) [**MVVM LIGHT**](http://dotnetpattern.com/tag/mvvm-light) [**INTERFACE**](http://dotnetpattern.com/tag/interface) [**GENERICS**](http://dotnetpattern.com/tag/generics) [**ARRAYLIST**](http://dotnetpattern.com/tag/arraylist) [**HASHTABLE**](http://dotnetpattern.com/tag/hashtable) [**USING**](http://dotnetpattern.com/tag/using) [**DESIGN PATTERNS**](http://dotnetpattern.com/tag/design-patterns) [**VISITOR**](http://dotnetpattern.com/tag/visitor) [**TEMPLATE**](http://dotnetpattern.com/tag/template) [**FLYWEIGHT**](http://dotnetpattern.com/tag/flyweight) [**STRATEGY**](http://dotnetpattern.com/tag/strategy) [**MEMENTO**](http://dotnetpattern.com/tag/memento) [**MEDIATOR**](http://dotnetpattern.com/tag/mediator) [**COMMAND**](http://dotnetpattern.com/tag/command) [**WCF**](http://dotnetpattern.com/tag/wcf) [**INSTANCE MANAGEMENT**](http://dotnetpattern.com/tag/instance-management) [**APPLICATION**](http://dotnetpattern.com/tag/application) [**ARCHITECTURE**](http://dotnetpattern.com/tag/architecture) [**AJAX**](http://dotnetpattern.com/tag/ajax) [**ANGULARJS**](http://dotnetpattern.com/tag/angularjs) [**FORMS**](http://dotnetpattern.com/tag/forms) [**VALIDATIONS**](http://dotnetpattern.com/tag/validations) [**NESTED FORMS**](http://dotnetpattern.com/tag/nested-forms) [**FILTERS**](http://dotnetpattern.com/tag/filters) [**CONTROLLERS**](http://dotnetpattern.com/tag/controllers) [**DIRECTIVES**](http://dotnetpattern.com/tag/directives) [**BINDINGS**](http://dotnetpattern.com/tag/bindings) [**EXPRESSIONS**](http://dotnetpattern.com/tag/expressions) [**NG-APP**](http://dotnetpattern.com/tag/ng-app) [**NG-MODEL**](http://dotnetpattern.com/tag/ng-model) [**ENVIRONMENT SETUP**](http://dotnetpattern.com/tag/environment-setup) [**THREADING**](http://dotnetpattern.com/tag/threading) [**SYNCHRONIZATION**](http://dotnetpattern.com/tag/synchronization) [**ENTITY FRAMEWORK**](http://dotnetpattern.com/tag/entity-framework) [**RHINO**](http://dotnetpattern.com/tag/rhino) [**MOCKS**](http://dotnetpattern.com/tag/mocks) [**TEST DOUBLES**](http://dotnetpattern.com/tag/test-doubles) [**OOPS**](http://dotnetpattern.com/tag/oops) [**APP**](http://dotnetpattern.com/tag/app) [**ANDROID**](http://dotnetpattern.com/tag/android) [**LINQ**](http://dotnetpattern.com/tag/linq) [**TESTING**](http://dotnetpattern.com/tag/testing) [**NUNIT**](http://dotnetpattern.com/tag/nunit) [**TESTCASE**](http://dotnetpattern.com/tag/testcase) [**TYPESCRIPT**](http://dotnetpattern.com/tag/typescript) [**JAVASCRIPT**](http://dotnetpattern.com/tag/javascript) [**ABSTRACT**](http://dotnetpattern.com/tag/abstract) [**NUMPY**](http://dotnetpattern.com/tag/NumPy)

# C# Hashtable

[C#](http://dotnetpattern.com/category/csharp)   [C#](http://dotnetpattern.com/tag/CSharp)   [hashtable](http://dotnetpattern.com/tag/hashtable)

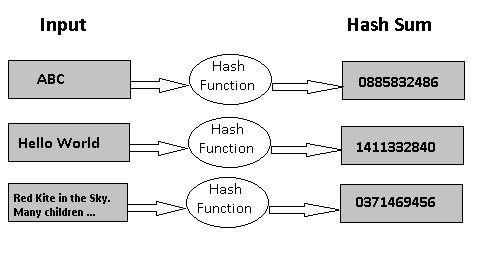
C# Hashtable class is a collection class where we can store data in the key/value pair. Hashtable is not type-safe. Hashtable is not generic and both key and value are based on object type.

Hashtable is used to store **unique keys** data. If we try to assign value to already used key, it just overrides the previous stored data.

It is based on the **Hashing algorithm** which is optimized for retrieving the data fast. It only takes O(1) operation to retrieve any value based on key.

Hashing algorithm is based on hash function. A hash function is any function which maps variable size key data to a particular fixed size data.

Below is the diagram of hashing algorithm:



It use hash sum to generate address to store data. When any request is received to retrieve data, it calculates the hash sum again using hash function and directly go to that address to retrieve the associated data.

**Below are some operations we can do on Hashtable class.**

## Add data

There are two ways to add data into Hashtable object:

1. Using Add method
2. Set key and value using Indexers

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Hashtable hashTable = new Hashtable();  hashTable.Add("1000", "James");  hashTable.Add("2000", "Michael");  hashTable.Add("3000", "Sandy");    Hashtable hashTableUsingIndexers = new Hashtable();  hashTableUsingIndexers["4000"] = "George";  hashTableUsingIndexers["5000"] = "Kapil";  hashTableUsingIndexers["6000"] = "Tod"; |

## Retrieve single entry

We can retrieve the data from Hashtable using indexers. Below is the example:

|  |  |
| --- | --- |
| 1 | string firstName = (string)hashTable["1000"]; //Result "James" |

As hashTable["1000"] returns object, we have to cast the result to string.

## Retrieve all entries

Hashtable implements the IEnumerable interface. That means we can use it in the foreach loop. In the loop, it returns the DictionaryEntry object each time we enumerate. DictionaryEntry class has two members:

1. Key: object
2. Value: object

We can use both members to retrieve the Key and Value of each entry. Below is the sample example:

|  |  |
| --- | --- |
| 1  2  3  4 | foreach(DictionaryEntry entry in hashTable)  {      Console.WriteLine("{0}-{1}", entry.Key, entry.Value);  } |

## Retrieve only Keys or Values

Hashtable provides a Keys and Values property to retrieve only the Keys or Values as collection.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | foreach(object key in hashTable.Keys)  {      Console.WriteLine(key.ToString());  }    foreach(object value in hashTable.Values)  {      Console.WriteLine(value.ToString());  } |

## Update entry in C# Hashtable

For updating an entry, we need to set value again using the same key. As I said earlier, when we assign value using the same key, it overrides the data. Below is the example:

|  |  |
| --- | --- |
| 1  2  3  4  5 | hashTable["3000"] = "Sandy";  //...  hashTable["3000"] = "Kapil";    string name = (string)hashTable["3000"]; //Returns Kapil |

## Remove entry

For removing an entry, we have to use Remove method in the Hashtable. Remove methods takes the Key as parameter. It removes the particular key/value pair from the records.

|  |  |
| --- | --- |
| 1 | hashTable.Remove("3000"); |

## Check Key/Value exists in Hashtable

There are three methods to find an entry. First two are based on searching on Keys and third one is based on using Value. All methods returns true if key/value pair exists else returns false.

1. Contains: Search using the Key
2. ContainsKey: Search using the Key
3. ContainsValue: Search using the Value

Below is the examples of all three methods:

|  |  |
| --- | --- |
| 1 | string searchByKey = "3000";  string searchByValue = "Michael";    if(hashTable.Contains(searchByKey))  {      // key exists  }    if(hashTable.ContainsKey(searchByKey))  {      // key exists  }    if(hashTable.ContainsValue(searchByValue))  {      // value exists  } |

## Get Total number of Items

To get the total number exists, we can use the Count property which returns the integer as total number of items.

|  |  |
| --- | --- |
|  | int totalRecords = hashTable.Count; |

## Final Words

Hashtable is optimized for faster retrieving the data. It computes the hash code from each key and use hash code to store the data. It only takes O(1) operation to retrieve any entry. Hashtable does not provide any type-safety.

Hashtable is an old data type to store data. Dictionary<> is a new data type to store key/Value pairs. Dictionary data type is of generic version. Dictionary provides type-safety and better performance as compared to Hashtable.

Below programs illustrate the use of above-discussed property:

**Example 1:**

|  |
| --- |
| // C# code to Gets or sets the value  // associated with the specified key  using System;  using System.Collections;    class GFG {        // Driver code      public static void Main()      {            // Creating a Hashtable          Hashtable myTable = new Hashtable();            // Adding elements in Hashtable          myTable.Add("g", "geeks");          myTable.Add("c", "c++");          myTable.Add("d", "data structures");          myTable.Add("q", "quiz");            // Get a collection of the keys.          ICollection c = myTable.Keys;            // Displaying the contents          foreach(string str in c)              Console.WriteLine(str + ": " + myTable[str]);            // Setting the value associated with key "c"          myTable["c"] = "C#";            Console.WriteLine("Updated Values:");            // Displaying the contents          foreach(string str in c)              Console.WriteLine(str + ": " + myTable[str]);      }  } |

**Output:**

d: data structures

c: c++

q: quiz

g: geeks

Updated Values:

d: data structures

c: C#

q: quiz

g: geeks

**Example 2:**

|  |
| --- |
| // C# code to Gets or sets the value  // associated with the specified key  using System;  using System.Collections;    class GFG {        // Driver code      public static void Main()      {            // Creating a Hashtable          Hashtable myTable = new Hashtable();            // Adding elements in Hashtable          myTable.Add("4", "Even");          myTable.Add("9", "Odd");          myTable.Add("5", "Odd and Prime");          myTable.Add("2", "Even and Prime");            // Get a collection of the keys.          ICollection c = myTable.Keys;            // Displaying the contents          foreach(string str in c)              Console.WriteLine(str + ": " + myTable[str]);            // Setting the value associated          // with key "56" which is not present          // will result in the creation of          // new key and value will be set which          // is given by the user          myTable["56"] = "New Value";            Console.WriteLine("Updated Values:");            // Displaying the contents          foreach(string str in c)              Console.WriteLine(str + ": " + myTable[str]);      }  } |

**Output:**

5: Odd and Prime

9: Odd

2: Even and Prime

4: Even

Updated Values:

5: Odd and Prime

9: Odd

2: Even and Prime

56: New Value

4: Even

System.Collections namespace in C# contains Hashtable that is quite similar to the Dictionary. Hashtable stores data in the form of key-value pairs.

It does that internally by assigning a hash code to hash key internally and whenever data is accessed it matches the hash code with a hash key to retrieve data. Each item in the table will have a key-value pair

How To Initialize A HashTable?

A HashTable can be initialized by using the keyword HashTable and then creating an instance of it.

Hashtable hashtbl = new Hashtable();

How To Add Elements To A HashTable?

Elements can be added to the HashTable using the method Add(). It allows the users to add an item with its key and value. It is not important to describe the data type of the key or value. While adding elements to the HashTable, you should remember that the keys cannot contain null whose values can be null.

HashTable.Add(Keys, Values);

Example:

class Program

{

static void Main(string[] args)

{

Hashtable hashtbl = new Hashtable();

hashtbl.Add("Number",1);

hashtbl.Add("Car", "Ferrari");

hashtbl.Add(8, "eight");

}

}

As you can see, we have added different datatypes as keys and values.

How To Access Elements Present In A HashTable?

One can retrieve the value of any key from the Hashtable using an indexer. But the indexer also requires a key to access and retrieve the value from the table.

Let’s add a small code snippet to the above program to fetch values:

class Program

{

static void Main(string[] args)

{

Hashtable hashtbl = new Hashtable();

hashtbl.Add("Number",1);

hashtbl.Add("Car", "Ferrari");

hashtbl.Add(8, "eight");

int value1 = (int)hashtbl["Number"];

String value2 = (string)hashtbl["Car"];

String value3 = (string)hashtbl[8];

Console.WriteLine(value1);

Console.WriteLine(value2);

Console.WriteLine(value3);

Console.ReadLine();

}

}

Now if we execute this, the following output will be produced:

1

ferrari

eight

In the above program, we have cast the value of all the keys for a given data type, to remove any compilation error that may occur. This is because being a non-generic collection, Hashtable doesn’t contain information about the data type of any of its contents i.e. keys and values.

Hence, if we directly try to get the data compiler, we will get confused about the datatype and will throw an error.

How To Remove An Element From Hashtable?

Let’s say we want to remove a particular key-value pair from the Hashtable. To achieve this we have to use Remove() method offered by the collections. Remove method deletes a given key-value pair from the Hashtable permanently.

Remove(Key);

Let’s add the Remove method in the above program to get the idea:

class Program

{

static void Main(string[] args)

{

Hashtable hashtbl = new Hashtable();

hashtbl.Add("Number",1);

hashtbl.Add("Car", "Ferrari");

hashtbl.Add(8, "eight");

int value1 = (int)hashtbl["Number"];

String value2 = (string)hashtbl["Car"];

String value3 = (string)hashtbl[8];

Console.WriteLine(value1);

Console.WriteLine(value2);

Console.WriteLine(value3);

//now we remove a key

hashtbl.Remove(8);

//Lets try to find the remove key now

String valueRemoved = (string)hashtbl[8];

Console.WriteLine("The value of the given key is :"+valueRemoved);

Console.ReadLine();

}

}

The output of the above program will be the following.

1

Ferrari

eight

The value of the given key is :

No, the value is printed to the console as we have removed the key from the Hashtable. Hence, the string value from valueRemoved is null.

If you want to remove everything from the hashtable, then C# provides us with another method called Clear(). Remove method deletes one key-value pair at a time whereas the Clear method deletes everything from the hashmap.

**# ArrayList**

ArrayList is a part of collections in C#. It is used to contain data of any given data type. It is similar to an array in C# but it doesn’t have a definite size. Its size increases automatically as more elements are added in it.

How To Initialize An ArrayList?

ArrayList can be initialized by using the “ArrayList” keyword.

ArrayList arrList = new ArrayList();

How To Add Elements Into An ArrayList?

You can add a single item into an array list by using the Add() method and a range of elements or multiple elements from another collection by using the AddRange() method.

**Example:**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.IO;  using System.Collections;  namespace ConsoleApp1  {      class Program      {          static void Main(string[] args)          {              ArrayList arrList = new ArrayList();              arrList.Add(7896);              arrList.Add("Seven");              Console.ReadLine();          }      }  } |

Here, you can see that we have used ArrayList.Add() method. As you can notice, we have added both integers as well as string in the same array list. This is possible as an array list is of non-generic data type, i.e. it can contain an element of any given data type.

How To Access An Element From An ArrayList?

An array list element can be accessed similar to an array i.e. by using the index. But as it’s a non-generic type, we first need to cast it to a suitable data type or need to use var keyword while accessing a value from the Array List.

**Example:**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.IO;  using System.Collections;  namespace ConsoleApp1  {      class Program      {          static void Main(string[] args)          {              ArrayList arrList = new ArrayList();              arrList.Add(7896);              arrList.Add("Seven");              //casted and retrieved data              int first\_index = (int)arrList[0];              string second\_index = (string)arrList[1];              Console.WriteLine("The first index value is : " + first\_index);              Console.WriteLine("The second index value is : " + second\_index);              Console.ReadLine();          }      }  } |

In the above example, we have retrieved the data by using the index of the array list and then we have cast it to the appropriate data type. The results have then been printed to the console as output.

**The output for the above program will be:**

**Output**

The first index value is: 7896  
The second index value is: Seven

How To Insert An Element Into ArrayList?

To insert an element in the ArrayList at a specified point or index. the Insert() method is used.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.IO;  using System.Collections;  namespace ConsoleApp1  {      class Program      {          static void Main(string[] args)          {              ArrayList arrList = new ArrayList();              arrList.Add(7896);              arrList.Add("Seven");              //casted and retrieved data              int first\_index = (int)arrList[0];              string second\_index = (string)arrList[1];              Console.WriteLine("The first index value is : " + first\_index);              Console.WriteLine("The second index value is : " + second\_index);              arrList.Insert(1, "Eight");              second\_index = (string)arrList[1];              Console.WriteLine("The second index value is : " + second\_index);              Console.ReadLine();          }      }  } |

Thus, we have inserted a new string at index 1 using the insert() method. So, if we run the above program we will get the following output:

**Output**

The first index value is: 7896  
The second index value is: Seven  
The second index value is: Eight

How To Remove An Element From ArrayList?

An element can be removed from the ArrayList using the Remove() method. Remove method accepts a parameter i.e. the value that needs to be removed from the array.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.IO;  using System.Collections;  namespace ConsoleApp1  {      class Program      {          static void Main(string[] args)          {              ArrayList arrList = new ArrayList();              arrList.Add(7896);              arrList.Add("Seven");              //casted and retrieved data              int first\_index = (int)arrList[0];              string second\_index = (string)arrList[1];              Console.WriteLine("The first index value is : " + first\_index);              Console.WriteLine("The second index value is : " + second\_index);              arrList.Insert(1, "Eight");              second\_index = (string)arrList[1];              Console.WriteLine("The second index value is : " + second\_index);              arrList.Remove(7896);              var data = arrList[0];              Console.WriteLine("The value at 0 index is : " + data);              Console.ReadLine();          }      }  } |

The remove method removes the given value from the list. When the value is removed from the given index, the subsequent value moves one index upward to fill the gap. As we are removing 0 indexes, the value from index 1 will move and fill the void at 0.

The output of the following program will be:

**Output**

The first index value is: 7896  
The second index value is: Seven  
The second index value is: Eight  
The value at 0 index is: Eight

How To Remove List Element Using The Index?

The user can remove a list element from a particular index by using the RemoveAt() method. RemoveAt() accepts a single parameter with an index number from which the element has to be removed. Similar to the Remove method, removing an element from the middle will push the next element to move one step upward to fill the gap.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.IO;  using System.Collections;  namespace ConsoleApp1  {      class Program      {          static void Main(string[] args)          {              ArrayList arrList = new ArrayList();              arrList.Add(7896);              arrList.Add("Seven");              //casted and retrieved data              int first\_index = (int)arrList[0];              string second\_index = (string)arrList[1];              Console.WriteLine("The first index value is : " + first\_index);              Console.WriteLine("The second index value is : " + second\_index);              arrList.Insert(1, "Eight");              second\_index = (string)arrList[1];              Console.WriteLine("The second index value is : " + second\_index);              arrList.RemoveAt(1);              var data = arrList[1];              Console.WriteLine("The value at 1 index is : " + data);              Console.ReadLine();          }      }  } |

In the above example, we have used, RemoveAt to remove index 1. Hence, the element at index 2 should move to index 1 to replace the gap.

The output of the following program will be:

**Output**

The first index value is : 7896  
The second index value is : Seven  
The second index value is : Eight  
The value at 1 index is : Seven

How To Sort And Reverse An ArrayList?

ArrayList offers two different methods for sorting and reverse operations. There is only one condition i.e. all the elements inside the array list should have the same data type for comparison with a comparator or else it will throw a runtime error.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.IO;  using System.Collections;  namespace ConsoleApp1  {      class Program      {          static void Main(string[] args)          {              ArrayList arrList = new ArrayList();              arrList.Add(7);              arrList.Add(4);              arrList.Add(5);              arrList.Add(1);              arrList.Add(3);              Console.WriteLine("Original Array List");              foreach (var v in arrList)              {                  Console.Write(v + " ");              }              //sorting an array list              Console.WriteLine();              Console.WriteLine("Sorted Array List");              arrList.Sort();              foreach (var srt in arrList)              {                  Console.Write(srt + " ");              }              //Reversing an array list              Console.WriteLine();              Console.WriteLine("Reversed Array List");              arrList.Reverse();              foreach (var rvrs in arrList)              {                  Console.Write(rvrs + " ");              }              Console.ReadLine();          }      }  } |

In the above program, we have first created an array list with integer data type and then we have used the sort method to sort the list and the reverse method to reverse the sorted list.

**So, the output of the following list will be:**

**Output**

Original Array List  
7 4 5 1 3  
Sorted Array List  
1 3 4 5 7  
Reversed Array List  
7 5 4 3 1

|  |
| --- |
| ArrayList arlist1 = new ArrayList() |
|  | { |
|  | 100, 200, 600 |
|  | }; |
|  |  |
|  | ArrayList arlist2 = new ArrayList() |
|  | { |
|  | 300, 400, 500 |
|  | }; |
|  | arlist1.InsertRange(2, arlist2); |
|  |  |
|  | foreach(var item in arlist1) |
|  | Console.Write(item + ", "); //output: 100, 200, 300, 400, 500, 600, |

**C# HashTable**

System.Collections namespace in C# contains Hashtable that is quite similar to the Dictionary. Hashtable stores data in the form of key-value pairs.

It does that internally by assigning a hash code to hash key internally and whenever data is accessed it matches the hash code with a hash key to retrieve data. Each item in the table will have a key-value pair

How To Initialize A HashTable?

A HashTable can be initialized by using the keyword HashTable and then creating an instance of it.

Hashtable hashtbl = new Hashtable();

How To Add Elements To A HashTable?

Elements can be added to the HashTable using the method Add(). It allows the users to add an item with its key and value. It is not important to describe the data type of the key or value. While adding elements to the HashTable, you should remember that the keys cannot contain null whose values can be null.

*HashTable.Add(Keys, Values);*

**Example:**

|  |
| --- |
| class Program      {          static void Main(string[] args)          {              Hashtable hashtbl = new Hashtable();              hashtbl.Add("Number",1);              hashtbl.Add("Car", "Ferrari");              hashtbl.Add(8, "eight");          }      } |

As you can see, we have added different datatypes as keys and values.

**How To Access Elements Present In A HashTable?**

One can retrieve the value of any key from the Hashtable using an indexer. But the indexer also requires a key to access and retrieve the value from the table.

**Let’s add a small code snippet to the above program to fetch values:**

|  |
| --- |
| class Program      {           static void Main(string[] args)          {              Hashtable hashtbl = new Hashtable();              hashtbl.Add("Number",1);              hashtbl.Add("Car", "Ferrari");              hashtbl.Add(8, "eight");              int value1 = (int)hashtbl["Number"];              String value2 = (string)hashtbl["Car"];              String value3 = (string)hashtbl[8];              Console.WriteLine(value1);              Console.WriteLine(value2);              Console.WriteLine(value3);              Console.ReadLine();          }      } |

**Now if we execute this, the following output will be produced:**

1  
ferrari  
eight

In the above program, we have cast the value of all the keys for a given data type, to remove any compilation error that may occur. This is because being a non-generic collection, Hashtable doesn’t contain information about the data type of any of its contents i.e. keys and values.

Hence, if we directly try to get the data compiler, we will get confused about the datatype and will throw an error.

## Add Items from another List

We can use the AddRange method for this work. AddRange takes the ICollection object as parameter. We have two ArrayList arr and arr2. As ArrayList implements the ICollection interface we can combine both lists using AddRange method.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | ArrayList arr = new ArrayList();  arr.Add("First");  arr.Add("Second");  arr.Add("Third");  arr.Add("Fourth");  arr.Add("Fifth");    ArrayList arr2 = new ArrayList();  arr2.Add("Sixth");  arr2.Add("Seventh");    arr.AddRange(arr2);    /// Items in ArrayList  /// First  /// Second  /// Third  /// Fourth  /// Fifth  /// Sixth  /// Seventh |

## Add multiple items at particular index

In the previous section, we have store only single item at a time at particular index. But if we need to store multiple items starting from particular index. Then we have to use a new method InsertRange method like shown below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | ArrayList arr = new ArrayList();  arr.Add("First");  arr.Add("Second");  arr.Add("Fifth");  arr.Add("Sixth");    ArrayList arr2 = new ArrayList();  arr2.Add("Third");  arr2.Add("Fourth");    arr.InsertRange(2, arr2);    foreach (object obj in arr)  {      Console.WriteLine(obj.ToString());  }    /// Result:  /// First  /// Second  /// Third  /// Fourth  /// Fifth  /// Sixth |

## Use ArrayList in multi-threaded scenarios

ArrayList is not thread-safe. Below is the example where two threads are accessing ArrayList. First thread is enumerating the list and second thread is adding item to the list.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | ArrayList arr = new ArrayList();  arr.Add("9");   // default item    var t1 = Task.Factory.StartNew(() =&amp;gt;      {          while (true)          {              Console.WriteLine("");              foreach (object obj in arr)              {                  Console.Write(obj.ToString() + ",");              }                if(arr.Count == 11)              {                  break;              }              Thread.Sleep(100);          }      });    var t2 = Task.Factory.StartNew(() =&amp;gt;      {          for (int i = 10; i &amp;lt; 20; ++i)          {              arr.Add(i);              Thread.Sleep(100);          }      });    Task.WaitAll(new Task[] { t1, t2 }); |

Above example fails with "Collection as modified, enumeration operation may not execute" error.

To use ArrayList in multi-threading environment, we must lock the ArrayList. For that when we enumerate the list, list should not be modified. ArrayList provides a SyncRoot object for synchronization. Below is the example of SyncRoot.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | ArrayList arr = new ArrayList();  arr.Add("9");   // default item    var t1 = Task.Factory.StartNew(() =>      {          while (true)          {              lock (arr.SyncRoot)              {                  Console.WriteLine("");                  foreach (object obj in arr)                  {                      Console.Write(obj.ToString() + ",");                  }              }              if (arr.Count == 11)              {                  break;              }              Thread.Sleep(100);          }      });    var t2 = Task.Factory.StartNew(() =>      {          for (int i = 10; i < 20; ++i)          {              lock (arr.SyncRoot)              {                  arr.Add(i);              }              Thread.Sleep(100);          }      });    Task.WaitAll(new Task[] { t1, t2 }); |

## Clear all items from the list

To clear all items, we have to use Clear method of list. Below is the example program:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | ArrayList arr = new ArrayList();  arr.Add("First");  arr.Add("Second");  arr.Add("Third");  arr.Add("Fourth");  arr.Add("Fifth");    arr.Clear();    int totalCount = arr.Count; //Return 0; |

How To Remove An Element From Hashtable?

Let’s say we want to remove a particular key-value pair from the Hashtable. To achieve this we have to use Remove() method offered by the collections. Remove method deletes a given key-value pair from the Hashtable permanently.

*Remove(Key);*

**Let’s add the Remove method in the above program to get the idea:**

|  |
| --- |
| class Program  {        static void Main(string[] args)      {          Hashtable hashtbl = new Hashtable();          hashtbl.Add("Number",1);          hashtbl.Add("Car", "Ferrari");          hashtbl.Add(8, "eight");          int value1 = (int)hashtbl["Number"];          String value2 = (string)hashtbl["Car"];          String value3 = (string)hashtbl[8];          Console.WriteLine(value1);          Console.WriteLine(value2);          Console.WriteLine(value3);          //now we remove a key          hashtbl.Remove(8);          //Lets try to find the remove key now          String valueRemoved = (string)hashtbl[8];          Console.WriteLine("The value of the given key is :"+valueRemoved);          Console.ReadLine();      }  } |

**The output of the above program will be the following.**

1  
Ferrari  
eight  
The value of the given key is :

No, the value is printed to the console as we have removed the key from the Hashtable. Hence, the string value from valueRemoved is null.

If you want to remove everything from the hashtable, then C# provides us with another method called Clear(). Remove method deletes one key-value pair at a time whereas the Clear method deletes everything from the hashmap.

**Let’s have a look at the below program to do this:**

|  |
| --- |
| class Program      {          static void Main(string[] args)          {              Hashtable hashtbl = new Hashtable();              hashtbl.Add("Number",1);              hashtbl.Add("Car", "Ferrari");              hashtbl.Add(8, "eight");              int value1 = (int)hashtbl["Number"];              String value2 = (string)hashtbl["Car"];              String value3 = (string)hashtbl[8];              Console.WriteLine(value1);              Console.WriteLine(value2);              Console.WriteLine(value3);              //now we remove a key              hashtbl.Remove(8);              //Lets try to find the remove key now              String valueRemoved = (string)hashtbl[8];              Console.WriteLine("The value of the given key is :"+valueRemoved);              // clearing all data from the HashTable              hashtbl.Clear();              Console.ReadLine();          }      } |

The above program will remove all the elements from the hashtable and will render it empty.

Other important methods offered by Hashtable is ContainsKey() and ContainsValue(). Both of these methods accept one argument which is a Key or a value and return a Boolean value. So, if the parameter passed with this method is present in the hashtable, then it will return a true value and if it’s not present then false will be returned.

**C# SortedList**

As the name suggests, SortedList contains sorted data in ascending order. It is similar to Hashtable as it contains similar key-value pair. All keys inserted or added to the SortedList are automatically arranged in ascending order.

How To Initialize A SortedList?

A SortedList can be initialized by using the keyword SortedList and creating an object instance for it.

SortedList sortedList = new SortedList();

The object can then be used for performing operations using the SortedList property and methods.

How To Add An Element To A SortedList?

You can add an element to a SortedList using the Add() method. A SortedList requires you to add a key and a value. You can add keys and values in any order you want and the sorted list will arrange all the added elements in ascending order.

|  |
| --- |
| class Program      {          static void Main(string[] args)          {              SortedList sortedList = new SortedList();              sortedList.Add(2, "Two Pineapples");              sortedList.Add(4, "Four Apples");              sortedList.Add(3, "Three Lemons");              sortedList.Add(5, "Five Mangoes");              sortedList.Add(1, "One Tree");              Console.ReadLine();          }      } |

In the above program, you can see that we have added integer as key and then string as the values. You can add any data type as you want and in any order you wish. The SortedList will arrange it in ascending order.

Similar to the HashTable, keys cannot be null and all the keys should have the same data type for comparison or else it will throw a compilation error.

The Sorted list sorts the element each time you add them. So, even if you add any element after the sorting has been done, it will again sort and add the element to its appropriate index.

How To Access The Elements From A SortedList?

A value in the Sorted List can be accessed by using the key.

**Let’s add a simple code to fetch the value from the SortedList described in the previous example:**

|  |
| --- |
| class Program      {          static void Main(string[] args)          {              SortedList sortedList = new SortedList();              sortedList.Add(2, "Two Pineapples");              sortedList.Add(4, "Four Apples");              sortedList.Add(3, "Three Lemons");              sortedList.Add(5, "Five Mangoes");              sortedList.Add(1, "One Tree");              string i = (string)sortedList[1];              string j = (string)sortedList[5];              string k = (string)sortedList[3];              Console.WriteLine(i);              Console.WriteLine(j);              Console.WriteLine(k);              Console.ReadLine();          }      } |

**The output of the above code snippet will be:**

One Tree  
Five Mangoes  
Three Lemons

In the above code snippet, we have to cast the value of all the keys for a given data type to remove any compilation error that may occur with the data type of the value. This is done to ensure that a compilation error is not thrown, even if some of the keys contain different data types.

How To Validate If A Given Key Exists In A SortedList?

There are two inbuilt methods i.e. *Contains()* and *ContainsKey()* that help us in determining if a particular key exists inside a sortedList. *ConstainsValue()* is another method that is used to determine if a particular value is present inside the collection or not.

**Let’s have a look at a simple program to learn more about these methods.**

|  |
| --- |
| class Program      {          static void Main(string[] args)          {              SortedList sortedList = new SortedList();              sortedList.Add(2, "Two Pineapples");              sortedList.Add(4, "Four Apples");              sortedList.Add(3, "Three Lemons");              sortedList.Add(5, "Five Mangoes");              sortedList.Add(1, "One Tree");              //boolean value for key 5              bool key = sortedList.ContainsKey(5);              //boolean value for vlaue "One Tree"              bool val = sortedList.ContainsValue("One Tree");              //Boolean value for unavailable key              bool unKey = sortedList.ContainsKey(25);              //Boolean value for unavailable value              bool unVal = sortedList.ContainsValue("some randome value");              Console.WriteLine("The sorted list contains 5 key :" + key);              Console.WriteLine("The sorted list contains One Tree value :" + val);              Console.WriteLine("The sorted list contains 25 key :" +unKey);              Console.WriteLine("The sorted list contains some random value:" + unVal);    Console.ReadLine();          }      } |

**The output of the above program will be:**

The sorted list contains 5 key : True  
The sorted list contains One Tree value : True  
The sorted list contains 25 key : False  
The sorted list contains some random value: False

In the above program, you can clearly see that if the value or key is present inside the Sorted List, then a true value is returned and if it is absent then a false value is returned.

**How To Remove An Element From The SortedList?**

The sorted list offers Remove() and RemoveAt() methods to delete any element present inside a Sorted List. Remove accepts a single argument with the key name and RemoveAt also accepts a single argument but with index.

Both of these methods remove any element present inside the Sorted list based on the argument.

**Let’s have a look at a simple code to understand it more clearly.**

|  |
| --- |
| class Program      {          static void Main(string[] args)          {              SortedList sortedList = new SortedList();              sortedList.Add(2, "Two Pineapples");              sortedList.Add(4, "Four Apples");              sortedList.Add(3, "Three Lemons");              sortedList.Add(5, "Five Mangoes");              sortedList.Add(1, "One Tree");              //The Remove() method accepts key as argument and removes both the key and the value              sortedList.Remove(1);              //Now we will check if the key is present              bool rmvKey = sortedList.ContainsKey(1);                Console.WriteLine("The presence if the key is: " + rmvKey);              //The RemoveAt() method acceots index as argument and remove any key and value present at that index              sortedList.RemoveAt(3);              Console.ReadLine();          }      } |

**The output of the above program will be:**

***The presence if the key is: False***

In the above program, we first used the Remove method to remove a key-value pair using the key. This will remove any key-value pair matching the key provided in the argument. Then we used the ContainsKey method to verify that the removed key no longer exists in the Sorted list.

In the next line, we used the RemoveAt method that removes elements using the index. Thus, as we discussed earlier once the particular element will be removed from the index, another element will move up to rearrange and maintain the sorted list structure.

**Conclusion**

Collections are dedicated classes present in the C# programming language to store and operate on data. These are used to perform specific actions i.e. to create dynamic lists, Reverse, Sort, etc. on a given data set.

In this tutorial, we learned about ArrayList which is similar to an array in some aspects but doesn’t have any predefined size. We also learned about HashTable that stores data in key-value pairs. Any value can be retrieved using the key.

We also learned about the sorted list which is similar to a HashTable but it automatically sorts all the data present inside it, in ascending order based on the keys.

The data inside the Sorted List is always in ascending order, i.e. even if you remove a particular element from the middle or add a new element to the Sorted List, it will automatically arrange all data in ascending order.

Sample Code

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using SeleniumFrameWork.FrameworkEssentials;  using SeleniumFrameWork.FrameWorkSupportModules;  namespace ConsoleApp1  {      class Program      {          static void Main(string[] args)          {               /\*             Array List Code             \*/              ArrayList arrList = new ArrayList();              arrList.Add(7);              arrList.Add(4);              arrList.Add(5);              arrList.Add(1);              arrList.Add(3);              Console.WriteLine("Original Array List");              foreach (var v in arrList)              {                     Console.Write(v + " ");              }              //sorting an array list              Console.WriteLine();              Console.WriteLine("Sorted Array List");              arrList.Sort();              foreach (var srt in arrList)              {                  Console.Write(srt + " ");              }              //Reversing an array list              Console.WriteLine();              Console.WriteLine("Reversed Array List");              arrList.Reverse();              foreach (var rvrs in arrList)              {                  Console.Write(rvrs + " ");              }              /\*                HashTable Code              \*/              Hashtable hashtbl = new Hashtable();              hashtbl.Add("Number", 1);              hashtbl.Add("Car", "Ferrari");              hashtbl.Add(8, "eight");              int value1 = (int)hashtbl["Number"];              String value2 = (string)hashtbl["Car"];              String value3 = (string)hashtbl[8];              Console.WriteLine(value1);              Console.WriteLine(value2);              Console.WriteLine(value3);              //now we remove a key              hashtbl.Remove(8);              //Lets try to find the remove key now              String valueRemoved = (string)hashtbl[8];              Console.WriteLine("The value of the given key is :" + valueRemoved);              // clearing all data from the HashTable              hashtbl.Clear();                /\*                Sorted List Code              \*/               SortedList sortedList = new SortedList();              sortedList.Add(2, "Two Pineapples");              sortedList.Add(4, "Four Apples");              sortedList.Add(3, "Three Lemons");              sortedList.Add(5, "Five Mangoes");              sortedList.Add(1, "One Tree");              string i = (string)sortedList[1];              string j = (string)sortedList[5];              string k = (string)sortedList[3];              Console.WriteLine(i);              Console.WriteLine(j);              Console.WriteLine(k);              //boolean value for key 5              bool key = sortedList.ContainsKey(5);              //boolean value for vlaue "One Tree"              bool val = sortedList.ContainsValue("One Tree");              //Boolean value for unavailable key              bool unKey = sortedList.ContainsKey(25);              //Boolean value for unavailable value              bool unVal = sortedList.ContainsValue("some randome value");              Console.WriteLine("The sorted list contains 5 key :" + key);              Console.WriteLine("The sorted list contains One Tree value :" + val);              Console.WriteLine("The sorted list contains 25 key :" +unKey);              Console.WriteLine("The sorted list contains some randome value:" + unVal);              //The Remove() method accepts key as argument and removes both the key and the value              sortedList.Remove(1);              //Now we will check if the key is present              bool rmvKey = sortedList.ContainsKey(1);              Console.WriteLine("The presence if the key is: " + rmvKey);              //The RemoveAt() method acceots index as argument and remove any key and value present at that index              sortedList.RemoveAt(3);              Console.ReadLine();          }      }  } |

Let’s have a look at the below program to do this:

class Program

{

static void Main(string[] args)

{

Hashtable hashtbl = new Hashtable();

hashtbl.Add("Number",1);

hashtbl.Add("Car", "Ferrari");

hashtbl.Add(8, "eight");

int value1 = (int)hashtbl["Number"];

String value2 = (string)hashtbl["Car"];

String value3 = (string)hashtbl[8];

Console.WriteLine(value1);

Console.WriteLine(value2);

Console.WriteLine(value3);

//now we remove a key

hashtbl.Remove(8);

//Lets try to find the remove key now

String valueRemoved = (string)hashtbl[8];

Console.WriteLine("The value of the given key is :"+valueRemoved);

// clearing all data from the HashTable

hashtbl.Clear();

Console.ReadLine();

}

}

The above program will remove all the elements from the hashtable and will render it empty.

Other important methods offered by Hashtable is ContainsKey() and ContainsValue(). Both of these methods accept one argument which is a Key or a value and return a Boolean value. So, if the parameter passed with this method is present in the hashtable, then it will return a true value and if it’s not present then false will be returned.

C# SortedList

As the name suggests, SortedList contains sorted data in ascending order. It is similar to Hashtable as it contains similar key-value pair. All keys inserted or added to the SortedList are automatically arranged in ascending order.

How To Initialize A SortedList?

A SortedList can be initialized by using the keyword SortedList and creating an object instance for it.

SortedList sortedList = new SortedList();

The object can then be used for performing operations using the SortedList property and methods.

How To Add An Element To A SortedList?

You can add an element to a SortedList using the Add() method. A SortedList requires you to add a key and a value. You can add keys and values in any order you want and the sorted list will arrange all the added elements in ascending order.

class Program

{

static void Main(string[] args)

{

SortedList sortedList = new SortedList();

sortedList.Add(2, "Two Pineapples");

sortedList.Add(4, "Four Apples");

sortedList.Add(3, "Three Lemons");

sortedList.Add(5, "Five Mangoes");

sortedList.Add(1, "One Tree");

Console.ReadLine();

}

}

In the above program, you can see that we have added integer as key and then string as the values. You can add any data type as you want and in any order you wish. The SortedList will arrange it in ascending order.

Similar to the HashTable, keys cannot be null and all the keys should have the same data type for comparison or else it will throw a compilation error.

The Sorted list sorts the element each time you add them. So, even if you add any element after the sorting has been done, it will again sort and add the element to its appropriate index.

How To Access The Elements From A SortedList?

A value in the Sorted List can be accessed by using the key.

Let’s add a simple code to fetch the value from the SortedList described in the previous example:

class Program

{

static void Main(string[] args)

{

SortedList sortedList = new SortedList();

sortedList.Add(2, "Two Pineapples");

sortedList.Add(4, "Four Apples");

sortedList.Add(3, "Three Lemons");

sortedList.Add(5, "Five Mangoes");

sortedList.Add(1, "One Tree");

string i = (string)sortedList[1];

string j = (string)sortedList[5];

string k = (string)sortedList[3];

Console.WriteLine(i);

Console.WriteLine(j);

Console.WriteLine(k);

Console.ReadLine();

}

}

The output of the above code snippet will be:

One Tree

Five Mangoes

Three Lemons

In the above code snippet, we have to cast the value of all the keys for a given data type to remove any compilation error that may occur with the data type of the value. This is done to ensure that a compilation error is not thrown, even if some of the keys contain different data types.

How To Validate If A Given Key Exists In A SortedList?

There are two inbuilt methods i.e. Contains() and ContainsKey() that help us in determining if a particular key exists inside a sortedList. ConstainsValue() is another method that is used to determine if a particular value is present inside the collection or not.

Let’s have a look at a simple program to learn more about these methods.

class Program

{

static void Main(string[] args)

{

SortedList sortedList = new SortedList();

sortedList.Add(2, "Two Pineapples");

sortedList.Add(4, "Four Apples");

sortedList.Add(3, "Three Lemons");

sortedList.Add(5, "Five Mangoes");

sortedList.Add(1, "One Tree");

//boolean value for key 5

bool key = sortedList.ContainsKey(5);

//boolean value for vlaue "One Tree"

bool val = sortedList.ContainsValue("One Tree");

//Boolean value for unavailable key

bool unKey = sortedList.ContainsKey(25);

//Boolean value for unavailable value

bool unVal = sortedList.ContainsValue("some randome value");

Console.WriteLine("The sorted list contains 5 key :" + key);

Console.WriteLine("The sorted list contains One Tree value :" + val);

Console.WriteLine("The sorted list contains 25 key :" +unKey);

Console.WriteLine("The sorted list contains some random value:" + unVal);

Console.ReadLine();

}

}

The output of the above program will be:

The sorted list contains 5 key : True

The sorted list contains One Tree value : True

The sorted list contains 25 key : False

The sorted list contains some random value: False

In the above program, you can clearly see that if the value or key is present inside the Sorted List, then a true value is returned and if it is absent then a false value is returned.

How To Remove An Element From The SortedList?

The sorted list offers Remove() and RemoveAt() methods to delete any element present inside a Sorted List. Remove accepts a single argument with the key name and RemoveAt also accepts a single argument but with index.

Both of these methods remove any element present inside the Sorted list based on the argument.

Let’s have a look at a simple code to understand it more clearly.

class Program

{

static void Main(string[] args)

{

SortedList sortedList = new SortedList();

sortedList.Add(2, "Two Pineapples");

sortedList.Add(4, "Four Apples");

sortedList.Add(3, "Three Lemons");

sortedList.Add(5, "Five Mangoes");

sortedList.Add(1, "One Tree");

//The Remove() method accepts key as argument and removes both the key and the value

sortedList.Remove(1);

//Now we will check if the key is present

bool rmvKey = sortedList.ContainsKey(1);

Console.WriteLine("The presence if the key is: " + rmvKey);

//The RemoveAt() method acceots index as argument and remove any key and value present at that index

sortedList.RemoveAt(3);

Console.ReadLine();

}

}

The output of the above program will be:

The presence if the key is: False

In the above program, we first used the Remove method to remove a key-value pair using the key. This will remove any key-value pair matching the key provided in the argument. Then we used the ContainsKey method to verify that the removed key no longer exists in the Sorted list.

In the next line, we used the RemoveAt method that removes elements using the index. Thus, as we discussed earlier once the particular element will be removed from the index, another element will move up to rearrange and maintain the sorted list structure.

Conclusion

Collections are dedicated classes present in the C# programming language to store and operate on data. These are used to perform specific actions i.e. to create dynamic lists, Reverse, Sort, etc. on a given data set.

In this tutorial, we learned about ArrayList which is similar to an array in some aspects but doesn’t have any predefined size. We also learned about HashTable that stores data in key-value pairs. Any value can be retrieved using the key.

We also learned about the sorted list which is similar to a HashTable but it automatically sorts all the data present inside it, in ascending order based on the keys.

The data inside the Sorted List is always in ascending order, i.e. even if you remove a particular element from the middle or add a new element to the Sorted List, it will automatically arrange all data in ascending order.

Sample Code

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using SeleniumFrameWork.FrameworkEssentials;

using SeleniumFrameWork.FrameWorkSupportModules;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

/\*

Array List Code

\*/

ArrayList arrList = new ArrayList();

arrList.Add(7);

arrList.Add(4);

arrList.Add(5);

arrList.Add(1);

arrList.Add(3);

Console.WriteLine("Original Array List");

foreach (var v in arrList)

{

Console.Write(v + " ");

}

//sorting an array list

Console.WriteLine();

Console.WriteLine("Sorted Array List");

arrList.Sort();

foreach (var srt in arrList)

{

Console.Write(srt + " ");

}

//Reversing an array list

Console.WriteLine();

Console.WriteLine("Reversed Array List");

arrList.Reverse();

foreach (var rvrs in arrList)

{

Console.Write(rvrs + " ");

}

/\*

HashTable Code

\*/

Hashtable hashtbl = new Hashtable();

hashtbl.Add("Number", 1);

hashtbl.Add("Car", "Ferrari");

hashtbl.Add(8, "eight");

int value1 = (int)hashtbl["Number"];

String value2 = (string)hashtbl["Car"];

String value3 = (string)hashtbl[8];

Console.WriteLine(value1);

Console.WriteLine(value2);

Console.WriteLine(value3);

//now we remove a key

hashtbl.Remove(8);

//Lets try to find the remove key now

String valueRemoved = (string)hashtbl[8];

Console.WriteLine("The value of the given key is :" + valueRemoved);

// clearing all data from the HashTable

hashtbl.Clear();

/\*

Sorted List Code

\*/

SortedList sortedList = new SortedList();

sortedList.Add(2, "Two Pineapples");

sortedList.Add(4, "Four Apples");

sortedList.Add(3, "Three Lemons");

sortedList.Add(5, "Five Mangoes");

sortedList.Add(1, "One Tree");

string i = (string)sortedList[1];

string j = (string)sortedList[5];

string k = (string)sortedList[3];

Console.WriteLine(i);

Console.WriteLine(j);

Console.WriteLine(k);

//boolean value for key 5

bool key = sortedList.ContainsKey(5);

//boolean value for vlaue "One Tree"

bool val = sortedList.ContainsValue("One Tree");

//Boolean value for unavailable key

bool unKey = sortedList.ContainsKey(25);

//Boolean value for unavailable value

bool unVal = sortedList.ContainsValue("some randome value");

Console.WriteLine("The sorted list contains 5 key :" + key);

Console.WriteLine("The sorted list contains One Tree value :" + val);

Console.WriteLine("The sorted list contains 25 key :" +unKey);

Console.WriteLine("The sorted list contains some randome value:" + unVal);

//The Remove() method accepts key as argument and removes both the key and the value

sortedList.Remove(1);

//Now we will check if the key is present

bool rmvKey = sortedList.ContainsKey(1);

Console.WriteLine("The presence if the key is: " + rmvKey);

//The RemoveAt() method acceots index as argument and remove any key and value present at that index

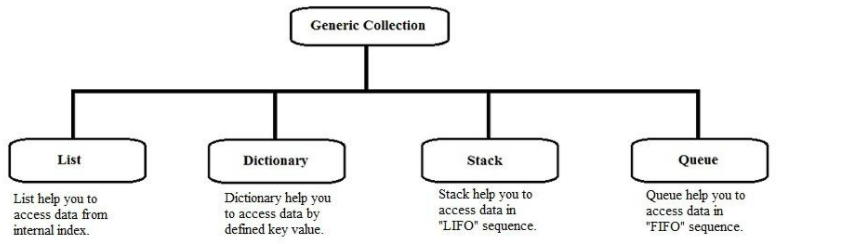
sortedList.RemoveAt(3);

Console.ReadLine();

}

}

}



Sample project with ArrayList

|  |
| --- |
| Practical Learning: Introducing the ArrayList Class |
|  |

1. Access the ListCreator.cs file. On the main menu, click File -> New
2. Type the following:

|  |
| --- |
| using System;  namespace Business  {  interface IStockItem  {  string PartNumber { get; set; }  string PartName { get; set; }  decimal UnitPrice { get; set; }  }  }  namespace AutoParts  {  public class Part : Business.IStockItem  {  private string ID;  protected string name;  protected decimal price;  private int qty;  public string PartNumber  {  get { return ID;}  set { ID = value; }  }  public string PartName  {  get { return name; }  set { name = value; }  }  public decimal UnitPrice  {  get { return (price < 0) ? 0.00M : price; }  set { price = value; }  }  public int Quantity  {  get { return (qty < 0) ? 0 : qty; }  set { qty = value; }  }  public Part()  {  this.ID = null;  this.name = "Unknown";  this.qty = 0;  this.price = 0.00M;  }  public Part(string Nbr, string nm, int q, decimal pr)  {  this.ID = Nbr;  this.name = nm;  this.qty = q;  this.price = pr;  }  }  } |

1. On the main menu, click File -> New
2. When asked whether you want to save the changes, click Yes
3. Inside of the CSharp Lessons folder, locate the **Libraries1** folder and display it in the Save In combo box
4. Change the Save As Type to All Files
5. Set the file name to **Parts2.cs** and click Save
6. Open the Command Prompt and change to the above Libraries1 folder
7. To create the DLL, type **csc /target:library /out:PartCreatorR2.dll Parts2.cs**and press Enter
8. In the new empty file of Notepad, type the following:

|  |
| --- |
| using System;  using System.Collections;  namespace AutoParts  {  class OrderProcessing  {  ArrayList ListOfParts;  public OrderProcessing()  {  ListOfParts = new ArrayList();  }  public void CreateInventory()  {  }  public void AddNewItem()  {  }  public void ShowInventory()  {  }  public void ProcessOrder()  {  }  public void DisplayReceipt()  {  }  }  } |

1. To save the file, on the main menu, click File -> Save
2. Locate the CSharp Lessons folder and display it in the Save In combo box
3. Create a new folder named **AutoParts2** and display it in the Save In combo box
4. Change the Save As Type to All Files. Set the File Name to **WorkOrder.cs**  and press Enter
5. Access the Exercise.cs file in the other instance of Notepad. On the main menu, click File -> New and type the following in it:

|  |
| --- |
| using System;  namespace AutoParts  {  class Exercise  {  static void Main()  {  OrderProcessing Order = new OrderProcessing();  }  }  } |

1. Save the new file as **Exercise.cs** in the same AutoParts2 folder
2. Open Windows Explorer or My Computer
3. From the CSharp Lessons\Libraries1 folder, copy ItemManagerR2.dll and paste it in the AutoParts2 folder

|  |
| --- |
| The Capacity of a List |
|  |

After declaring an **ArrayList** variable, it is empty. As objects are added to it, the list grows. The list can grow tremendously as you wish. The number of items of the list is managed through the memory it occupies and this memory grows as needed. The number of items that the memory allocated is currently using is represented by the **ArrayList.Capacity** property. This will usually be the least of your concerns.

If for some reason, you want to intervene and control the number of items that your **ArrayList** list can contain, you can manipulate the **Capacity** property. For example, you can assign it a constant to set the maximum value that the list can contain. Once, you will hardly have any reason to use the **Capacity** property: the compiler knows what to do with it.

If you set a fixed size on an **ArrayList** list, you may not be able to add a new item beyond the limit. In fact, if you attempt to do this, you may receive an error. A safe way is to check whether the list is fixed before performing a related operation. To find out whether a list is fixed, you can check the **ArrayList** variable's **IsFixedSize** property.

|  |
| --- |
| The Number of Items in the List |
|  |

When using a list, at any time, you should be able to know the number of items that the list contains. This information is provided by the **ArrayList.Count** property. The Capacity and the Count have this in common: the value of each increases as the list grows and the same value decreases if the list shrinks. It is important to know that, although they look alike, there are various differences between the capacity of a list and the number of items it contains. **Capacity** is a read/write property. This means that, as we saw above, you can assign a value to the capacity to fix the number of items that the list can contain. You can also retrieve the value of the Capacity. The **Count** is read-only because it is used by the compiler to count the current number of items of the items and this counting is performed without your intervention.

|  |
| --- |
| A Read-Only List |
|  |

One of the reason for creating a list is to be able to add items to it, edit its items, retrieve an items, or delete items from it. These are the default operations. You can still limit these operations as you judge them unnecessary. For example, you may create a list and then initialize it with the items that you want the list to only have. If you don't intend to have the user adding items to it, you can create the list as read-only. To do this, you can call the **ArrayList.ReadOnly()** method. It is overloaded with two versions as follows:

public static ArrayList ReadOnly(ArrayList);

public static IList ReadOnly(IList);

This method is static. This means that you don't need to declare an instance of **ArrayList** to call them. Instead, to make the list read-only, call the **ArrayList.ReadOnly()** method and pass your **ArrayList** variable to it.

As we will see in the next sections, some operations cannot be performed on a read-only list. To perform such operations, you can first find out whether an **ArrayList** list is read-only. This is done by checking its **IsReadOnly** property.

|  |
| --- |
| Item Addition |
|  |

The primary operation performed on a list is to create one. One of the biggest advantages of using a linked list is that you don't have to specify in advance the number of items of the list as done for an array. You can just start adding items. The **ArrayList** class makes this possible with the **Add()** method. Its syntax is:

public virtual int Add(object value);

The argument of this method is the value to add to the list. If the method succeeds with the addition, it returns the position where the value was added in the list. This is usually the last position in the list. If the method fails, the compiler would throw an error. One of the errors that could result from failure of this operation would be based on the fact that either a new item cannot be added to the list because the list is read-only, or the list was already full prior to adding the new item. Normally, a list can be full only if you had specified the maximum number of items it can contain using the **ArrayList.Capacity** property. As mentioned above, the list can be made read-only by passing its variable to the **ArrayList.ReadOnly()** method.

|  |
| --- |
| Practical Learning Practical Learning: Adding Items to an ArrayList List |
|  |

1. To create an inventory, access the WorkOrder.cs file and change it as follows:

|  |
| --- |
| using System;  using System.Collections;  namespace AutoParts  {  class OrderProcessing  {  ArrayList ListOfParts;  public OrderProcessing()  {  ListOfParts = new ArrayList();  }  // This method is used to create an inventory  public void CreateInventory()  {  Part One;    // Create a Part object using its properties  One = new Part();  One.PartNumber = "GD646";  One.PartName = "Bearing Clutch Pilot ";  One.UnitPrice = 9.75M;  One.Quantity = 4;  // Add the new part to the list  ListOfParts.Add(One);  // Create a Part object using a constructor  One = new Part("EU473", "Belt Accessory Drive ", 10, 6.75M);  // Add the new part to the list  ListOfParts.Add(One);  // Do the same to complete the list  One = new Part("AH325", "Break Drum ", 5, 20.55M);  ListOfParts.Add(One);  One = new Part("KS745", "Right Mirror ", 2, 9.35M);  ListOfParts.Add(One);  One = new Part("KE374", "Break Shoe ", 6, 20.25M);  ListOfParts.Add(One);  One = new Part("GD943", "Signal Lamp Assembly ", 4, 74.55M);  ListOfParts.Add(One);  One = new Part("GH386", "Bearing Input Shaft ", 3, 45.25M);  ListOfParts.Add(One);  One = new Part("WD394", "Brake Disc ", 14, 85.50M);  ListOfParts.Add(One);  One = new Part("TR944", "Front Wheel Lug Nut ", 7, 10.75M);  ListOfParts.Add(One);  One = new Part("GD844", "Front Pump Gasket ", 6, 10.72M);  ListOfParts.Add(One);  One = new Part("GD933", "Filter Steering ", 4, 12.55M);  ListOfParts.Add(One);  One = new Part("GW478", "Air Control Valve ", 8, 35.25M);  ListOfParts.Add(One);  One = new Part("LA943", "Clutch Master Clndr ", 5, 124.55M);  ListOfParts.Add(One);  One = new Part("RU688", "Tie Rod ", 12, 32.55M);  ListOfParts.Add(One);  One = new Part("PP797", "Ball Joint ", 14, 25.75M);  ListOfParts.Add(One);  One = new Part("RA292", "Drive Belt ", 10, 10.65M);  ListOfParts.Add(One);  One = new Part("AG778", "Oil Filter ", 8, 6.25M);  ListOfParts.Add(One);  One = new Part("KQ820", "Timing Belt ", 1, 45.95M);  ListOfParts.Add(One);  One = new Part("GT722", "Intake Manifold Gask ", 4, 18.55M);  ListOfParts.Add(One);  One = new Part("WA502", "Spark Plug Seal ", 24, 4.15M);  ListOfParts.Add(One);  One = new Part("AL848", "Air Filter ", 32, 15.65M);  ListOfParts.Add(One);  One = new Part("RU382", "Fuel Injector Clip ", 12, 17.05M);  ListOfParts.Add(One);  One = new Part("HJ624", "Brk Caliper w/o Pads ", 3, 190.50M);  ListOfParts.Add(One);  One = new Part("RL555", "Crankshaft Seal ", 7, 10.55M);  ListOfParts.Add(One);  One = new Part("PQ273", "Oil Pump ", 16, 218.75M);  ListOfParts.Add(One);  One = new Part("ER162", "Timing Belt Tensioner ", 12, 264.55M);  ListOfParts.Add(One);  One = new Part("EY275", "Camshaft Seal ", 8, 8.95M);  ListOfParts.Add(One);  One = new Part("LM357", "Valve Cover Gasket ", 1, 22.75M);  ListOfParts.Add(One);  One = new Part("RU473", "Valve Stem Seal ", 1, 3.95M);  ListOfParts.Add(One);  One = new Part("QW374", "Starter ", 1, 320.65M);  ListOfParts.Add(One);  One = new Part("QR374", "Radiator Cap ", 14, 12.75M);  ListOfParts.Add(One);  One = new Part("PQ902", "Thermostat Gasket ", 9, 4.20M);  ListOfParts.Add(One);  One = new Part("QT847", "Water Pump ", 5, 12.95M);  ListOfParts.Add(One);  One = new Part("PY784", "Spark Plug Platinum ", 14, 145.85M);  ListOfParts.Add(One);  One = new Part("TQ483", "Tie Rod Assembly ", 12, 3.95M);  ListOfParts.Add(One);  One = new Part("EQ173", "Oil Pump ", 20, 155.75M);  ListOfParts.Add(One);  One = new Part("UG376", "Piston Ring Set ", 13, 218.75M);  ListOfParts.Add(One);  One = new Part("PI489", "Distributor Cap ", 1, 275.55M);  ListOfParts.Add(One);  One = new Part("BT389", "Oil Seal Front Pump ", 18, 7.05M);  ListOfParts.Add(One);  One = new Part("CQ274", "Transmitter Filter Kit", 22, 9.25M);  ListOfParts.Add(One);  One = new Part("QX202", "Tail Lamp Assembly ", 7, 5.05M);  ListOfParts.Add(One);  One = new Part("GN780", "Bearing Wheel ", 5, 40.15M);  ListOfParts.Add(One);  One = new Part("XZ485", "Left Mirror ", 8, 7.25M);  ListOfParts.Add(One);  One = new Part("BD199", "Caliper Bolt/Pin ", 8, 3.55M);  ListOfParts.Add(One);  }  // This method is used to add a new part to the list  public void AddNewItem()  {  string ID;  string Name;  decimal Price;  int qty;  // Ask the user to type a number for the new part  Console.Write("Enter Item Number (Example: PD764): ");  ID = Console.ReadLine();  // Then ask the user to provide additional information about the part  Console.WriteLine("Enter the name or a short description: ");  Name = Console.ReadLine();  Console.Write("Enter Unit Price: ");  Price = decimal.Parse(Console.ReadLine());  Console.Write("How Many? ");  qty = int.Parse(Console.ReadLine());  // Using the new information that the user provided  // Create a new Part object using the second constructor  Part NewPart = new Part(ID, Name, qty, Price);    // Once the part is ready, add it to the database  ListOfParts.Add(NewPart);  }  public void ShowInventory()  {  }  public void ProcessOrder()  {  }  public void DisplayReceipt()  {  }  }  } |

1. Save the file
2. To test the above code, access the Exercise.cs file and change it as follows:

|  |
| --- |
| using System;  using System.Collections;  namespace AutoParts  {  class Exercise  {  static void Main()  {  int Choice = 0;  OrderProcessing Order = new OrderProcessing();  Order.CreateInventory();  // Display a short menu to the user before taking an action  try  {  Console.WriteLine(" =-= College Park Auto Parts =-=");  Console.WriteLine("How may I help you?");  Console.WriteLine("1. I want to process a customer's order");  Console.WriteLine("2. I want to see the current inventory");  Console.WriteLine("3. I want to add a new item to the inventory");  Console.Write("Your choice (1, 2, or 3)? ");  Choice = int.Parse(Console.ReadLine());  }  catch(FormatException)  {  Console.WriteLine("\nInvalid Choice - The program will terminate\n");  }  // Take an action based on the user's choice  switch(Choice)  {  case 1:  Order.ProcessOrder();  break;  case 2:  Order.ShowInventory();  break;  case 3:  Order.AddNewItem();  break;  }  }  }  } |

1. Save the file
2. Access the Command Prompt and change to the AutoParts2 folder
3. To compile the application, **csc /reference:PartCreatorR2.dll / out:"Four-Corner Auto-Parts R2".exe WorkOrder.cs Exercise.cs**and press Enter
4. To execute the application, type **"Four-Corner Auto-Parts R2"**and press Enter
5. After testing the program, return to Notepad

|  |
| --- |
| Item Retrieval |
|  |

Once a list is ready, you can perform different types of operations on it. Besides adding items, one of the most regular operations performed on a list consists of locating and retrieving its items. You have various options. To retrieve a single item based on its position, you can apply the square brackets of arrays to the variable. Like a normal array, an **ArrayList** list is zero-based. Another issue to keep in mind is that the **ArrayList**[] returns an Object value. Therefore, you may have to cast this value to your type of value to get it right.

|  |
| --- |
| Practical Learning Practical Learning: Retrieving Items From an ArrayList List |
|  |

1. To show an inventory, access the WorkOrder.cs file and change it as follows:

|  |
| --- |
| using System;  using System.Collections;  namespace AutoParts  {  class OrderProcessing  {  ArrayList ListOfParts;  public OrderProcessing()  {  ListOfParts = new ArrayList();  }  public void CreateInventory()  {  . . . *No Change*  }  public void AddNewItem()  {  string ID;  string Name;  decimal Price;  int qty;  Console.Write("Enter Item Number (Example: PD764): ");  ID = Console.ReadLine();  Console.WriteLine("Enter the name or a short description: ");  Name = Console.ReadLine();  Console.Write("Enter Unit Price: ");  Price = decimal.Parse(Console.ReadLine());  Console.Write("How Many? ");  qty = int.Parse(Console.ReadLine());  Part NewPart = new Part(ID, Name, qty, Price);  ListOfParts.Add(NewPart);  ShowInventory();  }  public void ShowInventory()  {  Console.WriteLine("\n===============================================");  Console.WriteLine("=-= College Park Auto Parts =-= Store Inventory");  Console.WriteLine("-----------------------------------------------");  Console.WriteLine(" Item # Description Price Qty");  for(int i = 0; i < ListOfParts.Count; i++)  {  Part One = (Part)ListOfParts[i];  Console.WriteLine(" {0} {1} {2,6}{3,5}",  One.PartNumber, One.PartName,  One.UnitPrice, One.Quantity);  }  Console.WriteLine("===============================================\n");  }  public void ProcessOrder()  {  ArrayList Choices = new ArrayList();  Part AnItem;  string PartID;  int Qty;    do  {  Console.Write("Enter the part number (q to stop): ");  PartID = Console.ReadLine();  for(int i = 0; i < ListOfParts.Count; i++)  {  AnItem = new Part();  if( PartID == ((Part)ListOfParts[i]).PartNumber)  {  AnItem.PartNumber = ((Part)ListOfParts[i]).PartNumber;  AnItem.PartName = ((Part)ListOfParts[i]).PartName;  AnItem.UnitPrice = ((Part)ListOfParts[i]).UnitPrice;    try  {  Console.Write("How many? ");  Qty = int.Parse(Console.ReadLine());  AnItem.Quantity = Qty;  }  catch(FormatException)  {  Console.WriteLine("Invalid Quantity!!!");  }  Choices.Add(AnItem);  break;  }  }  } while( PartID != "q" && PartID != "Q" );    DisplayReceipt(Choices);  }  public void DisplayReceipt(ArrayList lstItems)  {  decimal SubTotal = 0.00M,  TotalOrder = 0.00M;    Console.WriteLine("========================================================");  Console.WriteLine("=-= College Park Auto Parts =-= Receipt");  Console.WriteLine("------+---+-------------------------+-------+-----------");  Console.WriteLine("Part# Qty Description Price SubTotal");  Console.WriteLine("------+---+-------------------------+-------+-----------");    for(int i = 0; i < lstItems.Count; i++)  {  Part One = (Part)lstItems[i];  SubTotal = One.UnitPrice \* One.Quantity;  TotalOrder += SubTotal;  Console.WriteLine("{0} {1} {2} {3,6} {4,6}",  One.PartNumber, One.Quantity, One.PartName,  One.UnitPrice,SubTotal);  }    Console.WriteLine("------+---+-------------------------+-------+-----------");  Console.WriteLine("Total Order: {0:C}", TotalOrder);  Console.WriteLine("========================================================\n");  }  }  } |

1. Save, compile, and test the program. Here is an example:

|  |
| --- |
| =-= College Park Auto Parts =-=  How may I help you?  1. I want to process a customer's order  2. I want to see the current inventory  3. I want to add a new item to the inventory  Your choice (1, 2, or 3)? 1  Enter the part number (q to stop): GN780  How many? 1  Enter the part number (q to stop): CQ274  How many? 1  Enter the part number (q to stop): RL555  How many? 2  Enter the part number (q to stop): WA502  How many? 2  Enter the part number (q to stop): PY784  How many? 1  Enter the part number (q to stop): q  ========================================================  =-= College Park Auto Parts =-= Receipt  ------+---+-------------------------+-------+-----------  Part# Qty Description Price SubTotal  ------+---+-------------------------+-------+-----------  GN780 1 Bearing Wheel 40.15 40.15  CQ274 1 Transmitter Filter Kit 9.25 9.25  RL555 2 Crankshaft Seal 10.55 21.10  WA502 2 Spark Plug Seal 4.15 8.30  PY784 1 Spark Plug Platinum 145.85 145.85  ------+---+-------------------------+-------+-----------  Total Order: $224.65  ======================================================== |

|  |
| --- |
| Item Location |
|  |

Instead of the square brackets that allow you to retrieve an item based on its position, you can look for an item based on its complete definition. You have various options. You can first "build" an item and ask the compiler to check whether any item in the list matches your definition. To perform this search, you can call the **ArrayList.Contains()** method. Its syntax is:

public virtual bool Contains(object item);

The item to look for is passed as argument to the method. The compiler would look for exactly the item, using its definition, in the list. If any detail of the argument fails to match any item of the **ArrayList** list, the method would return false. If all characteristics of the argument correspond to an item of the list, the method returns true.

Another option to look for an item in a list consists of calling the **ArrayList.BinarySearch()** method. It is overloaded in three versions and one of them uses the following syntax:

public virtual int BinarySearch(object value);

The item to look for is passed argument to the method.

|  |
| --- |
| Item Deletion |
|  |

As opposed to adding an item to a list, you may want to remove one. To perform this operation, you have various options. You can ask the compiler to look for an item in the list and if, or once, the compile finds it, it would delete the item. To perform this type of deletion, you can call the **ArrayList.Remove()** method. Its syntax is:

public virtual void Remove(object obj);

This method accepts as argument the item that you want to delete from the list. To perform this operation, the list must not be read-only.

The **Remove()** method allows you to specify the exact item you want to delete from a list. Another option you have consists of deleting an item based on its position. This is done using the **RemoveAt()** method whose syntax is:

public virtual void RemoveAt(int index);

With this method, the position of the item is passed as argument. If the position is not valid because either it is lower or higher than the current **Count**, the compiler would throw an **ArgumentOutOfRangeException** exception.

To remove all items from a list at once, you can call the **ArrayList.Clear()** method. Its syntax is:

public virtual void Clear();