# Arrays in C#

آرایه مجموعه یا collections از یک نوع داده است که مکانی از حافظه تحت یک نام آن ها را نگه داری می کند.

آرایه نوعی ساختار داده ترتیبی است sequential data structure که مجموعه ای داده های یک نوع را در خود نگه داری می کند.

**How does using [] this work in real memory?**

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

زمانی که از علامت [] استفاده می کنید به کامپایلر می گویید که نوع داده ی که در حال استفاده از آن هستید یک آرایه است و بلوکی از حافظه به آن تخصیص داده شود.

**Types of Arrays in C#:**

1. **Single dimensional array تک بعدی**
2. **Multi-dimensional array چند بعدی**

در آرایه تک بعدی دیتا به صورت row یا ردیف در کنار هم قرار می گیرند در حالی که در آرایه های چند بعدی داده های به صورت ردیف و ستون rows and columns در کنار هم قرار می گیرند.

آرایه های چند بعدی 2 نوع هستند

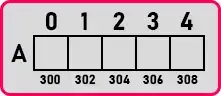
1. **Jagged array**: Whose rows and columns are not equal دندانه دار ردیف و ستون نا برابر
2. **Rectangular array**: Whose rows and columns are equal مستطیلی ردیف و ستون ها برابر

**نکته:** برای دسترسی به آرایه می توان از index استفاده کرده و همیشه اولین خانه دارای مقدار برابر است با صفر 0 و تعداد داده های آرایه نیز برابر است با تعداد خانه ها -1

ساختن آرایه با 5 خانه

**Example: int[] A = new int[5];**

فرض کنید خانه حافظه به شکل زیر است از 300 شروع شده و آرایه ما از نوع int است که هر کدام 2byte فضا می گیرد پس حافظه به شکل زیر خواهد بود.



در این مثال خانه a[0] برابر است با 300 و 3001 , a[1] برابر است با 302 و 303

Indexer

در آرایه در واقع یک pointer است که برای دسترسی به داده آرایه در یک خانه مشخص استفاده می شود و از 0 تا n-1 ادامه دارد.

Array is fast?

به دلیل این که مقادیر در حافظه به صورت پیوسته یا پشت سر هم قراردارند دسترسی به آن ها سریع است.

**2 نوع آرایه در سی شارپ داریم طول ثابت و متغییر در طول متغییر سایز آرایه به صورت خودکار با افزایش ایتم های جدید اضافه می شود.**

**نکته: array is reference type and heap memory is allocate**

**چرا می توان روی آرایه foreach زد؟**

به دلیل پیاده سازی رابط IEnumerable می توانیم بر روی عناصر آرایه پیمایش کنیم

Int[] arr = {1,23,4,5,6} روش initialize کردن آرایه.

Foreach

به صورت مخصوص برای دسترسی به عناصر آرایه ها ایجاد شده است و از عنصر اول تا آخر بدون نیاز به شرط پیمایش می کند.

Foreach vs for

For نیاز به دسترسی به index دارد ولی foreach مستقیما به داده دسترسی دارد

نکته : for برای دسترسی و assign کردن کاربرد دارد ولی foreach فقط برای دسترسی به داده.

نکته: دسترسی به عنصری از آرایه که خارج از محدوده است سبب exception می شود

**Array base class is implementing IList interface**

# 2D Arrays in C#

آرایه های که در سی شارپ که عناصر آن در ردیف و ستون rows & column ذخیره می شوند دو بعدی نام دارند.

انوع آرایه 2 بعدی

Rectangular در این نوع آرایه ها تعداد ردیف ها و ستون ها با هم برابر است

Jagged در این نوع آرایه ها تعداد ردیف ها و ستون ها با هم برابر نیست.

در این نوع آرایه ها به هر عنصر از آرایه با 2 index دسترسی می توان داشت المان ها در این نوع آرایه ها به صورت matrix ذخیره می شوند.

در این نوع آرایه ها index اول به ردیف row و index دوم به column یا ستون اشاره می کند.

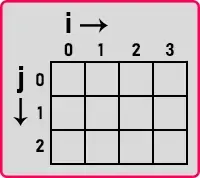
**int[,] matrix = new int[3,3];**



برای دسترسی به خانه ی اول matrix[0][0] به این روش عمل می کنیم.

**How 2D Array is Created and Accessed in C#?**

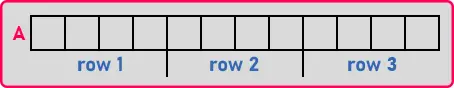
**int[,] A = new int[3,4];**



**برای دسترسی و چاپ عنصر آرایه از روش زیر استفاده می کنیم.**

**Console.WriteLine(A[1,2]);**

نحوه ذخیره سازی و دسترسی به عناصر آرایه 2 بعدی در حافظه Ram به این صورت است که در ردیف های پشت سر هم در حافظه داده ها ذخیره می شوند و تقسیم بندی منطقی ردیف ها انجام می شود



نحوه تعریف یک آرایه با 2 ردیف داده و 3 ستون

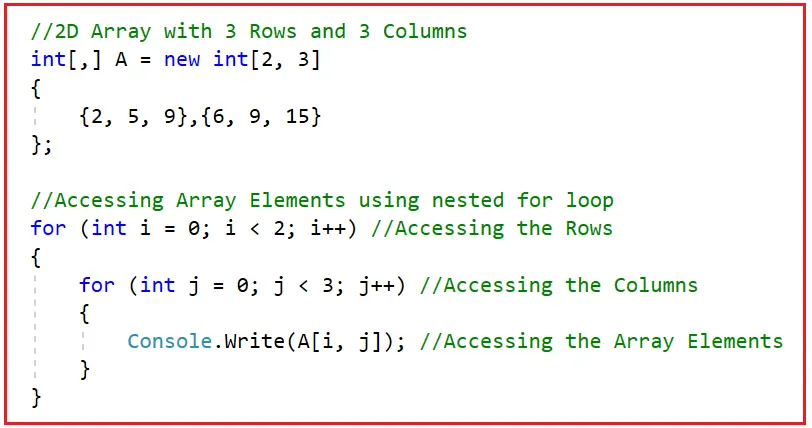
**int[,] A = new int[2,3]**  
**{**  
**{2, 5, 9},{6, 9, 15}**  
**};**

**OR**

**int[,] A = {{2, 5, 9},{6, 9, 15}};**

**Accessing the Elements of the 2D array in C#:**

برای دسترسی به عناصر آرایه های 2 بعدی به 2 حلقه تو در تو نیاز داریم یکی برای دسترسی به عناصر row و دیگری برای دسترسی به عناصر columns به مثال زیر توجه کنید.



**نکته** برای دسترسی به تعداد ردیف ها GetLength(0) و برای دسترسی به تعداد ستون ها GetLenght(1) استفاده می شود.

مثال پرکردن عناصر آرایه:

//Assigning values to the 2D array by using nested for loop

//arr.GetLength(0): Returns the size of the Row

//arr.GetLength(0): Returns the size of the Column

**for** **(int** i = 0; i **<** RectangleArray.GetLength**(**0**)**; i++**)**

**{**

**for** **(int** j = 0; j **<** RectangleArray.GetLength**(**1**)**; j++**)**

**{**

a += 5;

RectangleArray**[**i, j**]** = a;

**}**

**}**

**Jagged Array in C#:**

نوع دیگری از آرایه های 2 بعدی هستند که شباهتی بسیاری به آرایه های 2 بعدی مستطیلی دارد با این تفاوت که در این نوع آرایه ها می توان هر ردیف داده دارای تعداد ستون داده متفاوتی باشد مثلا ردیف اول دارای 4 ستون داده و ردیف دوم دارای 2 ستون داده باشد

به این نوع آرایه، آرایه ی از آرایه ها نیز می گویند به این دلیل که هر ردیف خود یک آرایه تک بعدی است.

در تعریف این نوع آرایه ها فقط کافی است تعداد row را مشخص کنید. و سپس برای هر row می توانید تعداد ستون ها را مشخص کنید.

**int [][] arr = new int[4][];**

**arr[0] = new int[5];** // we want five columns in the first row  
**arr[1] = new int[6];** // we want six columns in the first row  
**arr[2] = new int[4];** // we want four columns in the first row  
**arr[3] = new int[5];** // we want five columns in the first row

در ادامه نحوه دسترسی به عناصر آرایه را بررسی می کنیم.

//Printing the values of Jagged array using nested for loop

//It will print the default values as we are not assigning any

//values to the array

//GetLength(0): Returns the Size of the Rows (4)

Console.WriteLine**(**"Printing the Default Values of Jagged Array:"**)**;

**for** **(int** i = 0; i **<** arr.GetLength**(**0**)**; i++**)**

**{**

//arr[i].Length: Returns the Length of Each Row

**for** **(int** j = 0; j **<** arr**[**i**]**.Length; j++**)**

**{**

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

**}**

**}**

# Advantages and Disadvantages of Arrays in C#

مزایا Advantages

ذخیره کردن چندین مقدار از یک نوع تحت عنوان یک نام

برای پیاده سازی سایر ساختار های داده مانند linked list graphs stacks queues کاربرد دارد

آرایه های 2 بعدی برای پیاده سازی ماتریکس کاربرد دارند.

از نوع strongly type هستند.

چون از نوع strongly هستند مزایای دارند :

Performance اپلیکیشن ما بهتر می شود چون boxing , unboxing رخ نمی دهد و خطای RunTime رخ نمی دهد چون در زمان compile متوجه خطا ها می شویم.

**نکته : loosely type collection مثل ArrayList نوع های کالکشنی هستند که می توان در آن ها هر نوع داده ی را نگه داری کرد و در زمان کامپایل خطا نمی دهد.**

**معایت Disadvantages**

Fix size به معنی این است که زمانی که ارایه ایجاد می شود دیگر قابلیت increase or decrease برای آن آرایه وجود ندارد

اگر حافظه ی بیشتر از نیاز خود اختصاص دهیم waste می شود حافظه و اگر کمتر از نیاز تخصیص دهیم دچار مشکل می شویم

هیچ وقت نمی توانیم که مقداری را از middle of array اضافه کنیم یا همچنین نمی توان از وسط آرایه حذف کرد.

# Collections in C#

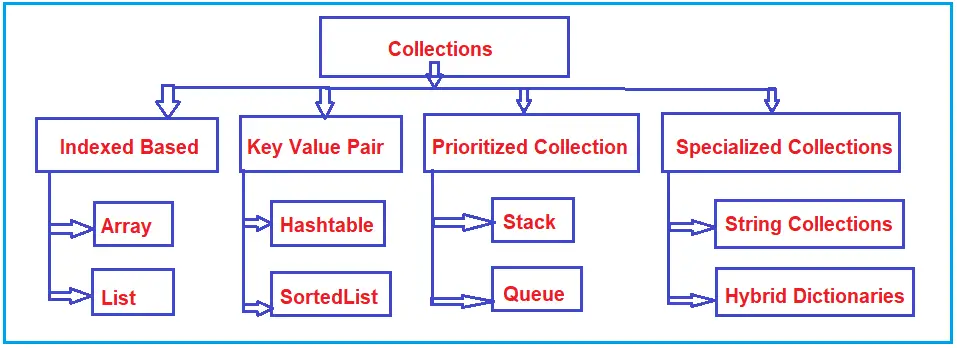
کالکشن ها شبیه آرایه هستند و راهی منعطف برای کار کردن با گروهی از اشیا هستند.

گروهی از رکورد ها هستند که می توانند تحت یک واحد منطقی استفاده شوند.

Categories of Collections

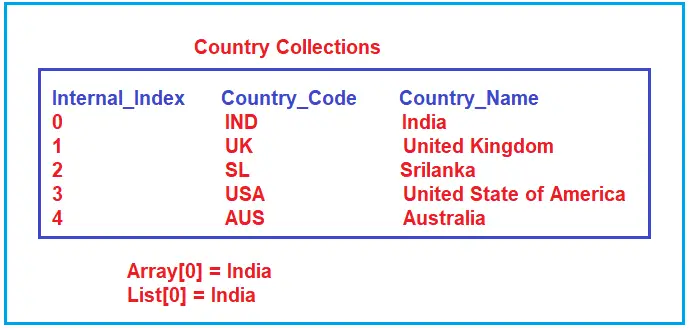
4 نوع کلی هستند که عبارتند از

Indexed – kay value pair – prioritized collection – specialized collections



**Indexed Base Collections:**

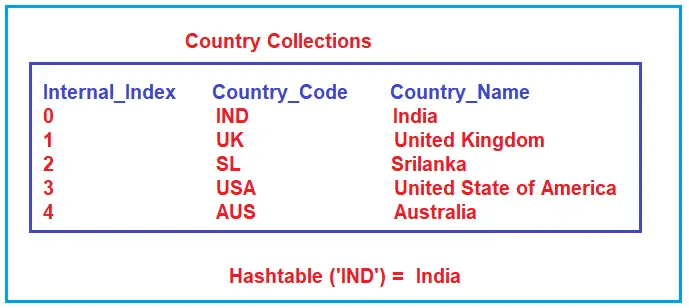
کالکشنی مانند Array or List را در نظر بگیرید این نوع ها زمانی که المانی به آن ها اضافه می شود به صورت خودکار توسط فریم ورک برای آن عنصر یک index اختصاص داده می شود که از طریق این index اطلاعات ما قابل بازیابی است.



**Key-Value Pair Collections**

از انواع آن ها HashTable – Dictionary – SortedList هستند

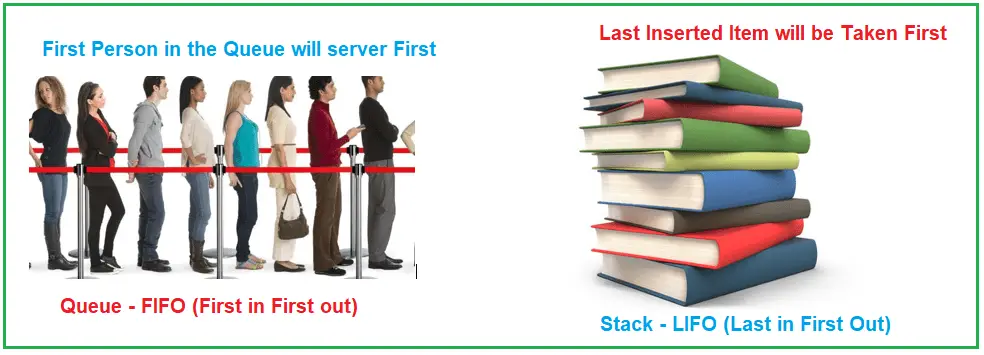
در پروژه ها بزرگ معمولا برای identify کردن رکورد ها از index استفاده نمی کنیم بلکه از یک key استفاده می شود که در ادامه مثال آن را مشاهده خواهید کرد. برای بازیابی رکورد ها با استفاده از کلید این ها مناسب هستند.



Prioritized Collections:

مجموعه های اولویت دار.

کمک می کند که به عناصر به ترتیب خاصی دسترسی داشت. Stack , queue از دسته بندی کالکشن های اولیت دار است مثلا برای پیاده سازی FIFO می توان از Queue استفاده کرد و برای پیاده سازی LIFO می توان از نوع Stack استفاده کرد.



**Specialized Collections:**

The Specialized Collections are specifically designed for a specific purpose. For example, a Hybrid Dictionary starts as a list and then becomes a hashtable.

**What are Array and Their disadvantages in C#?**

به طور مثال در آرایه ها لازم است که شما ابتدا size آرایه را مشخص کنید و در صورتی که در طول اجرای برنامه نیاز به تغییر size آرایه باشد این امکان فراهم نیست نیاز است به ایجاد یک آرایه به صورت دستی و کپی کردم مقادیر به آن است.

عدم امکان increase or decrease کردن سایز آرایه و همچنین افزودن یا کاستن مقدار از middle of array.

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

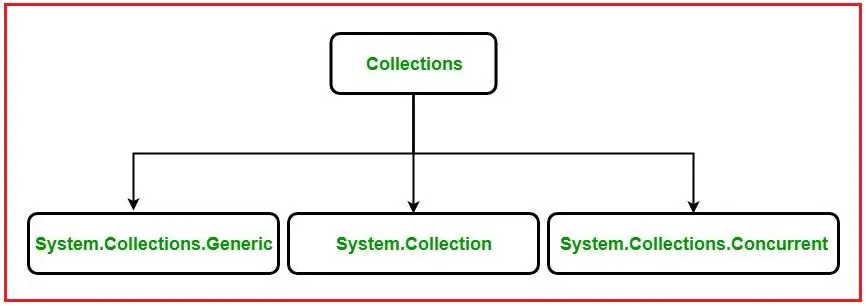
کلاس های در فضای نام system.Collections هستند که دارای ظرفیت و عملکرد بهتر نسبت به آرایه های سنتی هستند. Collections در سی شارپ reusable هستند و کارآمدی بهتر دارند.

به زبان ساده کالکشن ها در سی شارپ برابر dynamic array است.

قابلیت افزایش سایز به صورت پویا

درج کردم مقدار جدید در middle of array و همچنین حذف مقدار. قابلیت مرتب سازی جستجو بروزرسانی و حذف و .. را دارند.

**Types of Collections in C#**



Non generic برای مدیرت هر نوع داده ی کاربرد دارد. بر روی object کار می کند.

1. [**ArrayList**](https://dotnettutorials.net/lesson/arraylist-collection-csharp/)**:** It Implements the System.Collections.IList interface using an array whose size is dynamically increased as required.
2. [**Stack**](https://dotnettutorials.net/lesson/stack-collection-csharp/)**:** It represents a simple last-in-first-out (LIFO) non-generic collection of objects.
3. [**Queue**](https://dotnettutorials.net/lesson/queue-collection-class-csharp/)**:** It represents a first-in, first-out collection of objects.
4. [**Hashtable**](https://dotnettutorials.net/lesson/hashtable-csharp/)**:** It represents a collection of key/value pairs that are organized based on the hash code of the key.
5. [**SortedList**](https://dotnettutorials.net/lesson/sortedlist-collection-class-in-csharp/)**:** It represents a collection of key/value pairs that are sorted by the keys and are accessible by key and by index.

**Generic Collections Classes in C#:**

این نوع به صورت type safe هستند یعنی فقط مواردی ار که با نوع مجموعه compatible است را می توان در آن ها ذخیره کرد.

1. [**List<T>**](https://dotnettutorials.net/lesson/list-collection-csharp/)**:** It represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists.
2. [**Stack<T>**](https://dotnettutorials.net/lesson/generic-stack-csharp/)**:** It represents a variable size last-in-first-out (LIFO) collection of instances of the same specified type.
3. [**Queue<T>**](https://dotnettutorials.net/lesson/generic-queue-collection-class-csharp/)**:** It represents a first-in, first-out collection of objects.
4. [**HashSet<T>**](https://dotnettutorials.net/lesson/generic-hashset-collection-class-in-csharp/)**:** It represents a set of values. It removes duplicate elements from the collection.
5. [**Dictionary<TKey, TValue>**](https://dotnettutorials.net/lesson/dictionary-generic-collection-csharp/)**:** It represents a collection of keys and values.
6. [**SortedList<TKey, TValue>**](https://dotnettutorials.net/lesson/generic-sortedlist-collection-class-in-csharp/)**:** It represents a collection of key/value pairs that are sorted by key based on the associated System.Collections.Generic.IComparer implementation.
7. [**SortedSet<T>**](https://dotnettutorials.net/lesson/generic-sortedset-collection-class-in-csharp/)**:** It represents a collection of objects that are maintained in sorted order.
8. [**SortedDictionary<TKey, TValue>**](https://dotnettutorials.net/lesson/generic-sorteddictionary-collection-class-in-csharp/)**:** It represents a collection of key/value pairs that are sorted on the key.
9. [**LinkedList<T>**](https://dotnettutorials.net/lesson/generic-linkedlist-collection-class-in-csharp/)**:** It represents a doubly linked list.

**Concurrent Collection Classes in C#:**

این نوع ها safe collection هستند برای کار با thread ها زمانی که چندین thread به صورت همزمان به collection دسترسی دارند فضای نام Concurrent کلاس های را برای عملیات Thread – safe فراهم می کند که مشکلی برای دسترسی به ایتم های کالکشن ایجاد نشود.

1. [**BlockingCollection<T>**](https://dotnettutorials.net/lesson/blockingcollection-class-in-csharp/)**:** It provides blocking and bounding capabilities for thread-safe collections that implement System.Collections.Concurrent.IProducerConsumerCollection.
2. [**ConcurrentBag<T>**](https://dotnettutorials.net/lesson/concurrentbag-collection-class-in-csharp/)**:** It represents a thread-safe, unordered collection of objects.
3. [**ConcurrentStack<T>**](https://dotnettutorials.net/lesson/concurrentstack-collection-class-csharp/)**:** It represents a thread-safe last-in-first-out (LIFO) collection.
4. [**ConcurrentQueue<T>**](https://dotnettutorials.net/lesson/concurrentqueue-collection-class-in-csharp/)**:** It represents a thread-safe first-in-first-out (FIFO) collection.
5. [**ConcurrentDictionary<TKey, TValue>**](https://dotnettutorials.net/lesson/concurrentdictionary-collection-class-in-csharp/)**:** It represents a thread-safe collection of key/value pairs that can be accessed by multiple threads concurrently.

# ArrayList in C#

از نوع کالکشن های non-generic است مانند یک Array است اما امکاناتی مانند dynamic resize , adding deleting از میانه کالکشن را فراهم می کند.

می تواند برای افزودن unknow data استفاده شود زمانی که type , size داده را نمی دانیم کاربرد دارد.

ویژگی مهم آن افرایش سایز و استفاده از نوع داده یکسان و غیر یکسان در این نوع کالکشن ها است.

ظرفیت arrayList برابر است با تعداد عناصر موجود در آن. قابلیت عنصر تکراری دارد و می تواند مقادیر duplicate را نیز نگه داری کند.

**How to Create an ArrayList in C#?**

1. **ArrayList():** The method is used to initialize a new instance of the ArrayList class that is empty and has the default initial capacity.
2. **ArrayList(ICollection c):** The method is used to initialize a new instance of the ArrayList class that contains elements copied from the specified collection and that have the same initial capacity as the number of elements copied. The parameter c specifies the Collection whose elements are copied to the new list.
3. **ArrayList(int capacity):** The method is used to initialize a new instance of the ArrayList class that is empty and has the specified initial capacity. The parameter capacity specifies the number of elements that the new list can initially store.

نکته: در یک ArrayList امکان اضافه کردن نوع های داده مختلف وجود دارد. قابلیت object initializer نیز دارد

var arrayList2 = new ArrayList**()**

**{**

102, "Smith", "Smith", **true**, 15.6

**}**;

**How to Access an ArrayList in C#?**

اگر definition کلاس ArrayList را نگاه کنید می بینید که رابط IList را پیاده سازی کرده است به همین دلیل با استفاده از **indexer به عناصر آن دسترسی داریم نکته IList**

**نکته:** زمانی که داده ی در ArrayList ذخیره می شود به صورت خدکار به object تغییر پیدا می کند برای بازیابی داده یا باید از نوع var استفاده کرد یا از Explicit cast استفاده کنید.

//Accessing individual elements from ArrayList using Indexer

**int** firstElement = **(int)**arrayList1**[**0**]**; //returns 101

**string** secondElement = **(string)**arrayList1**[**1**]**; //returns "James"

//int secondElement = (int) arrayList1[1]; //Error: cannot cover string to int

Console.WriteLine**(**$"First Element: {firstElement}, Second Element: {secondElement}"**)**;

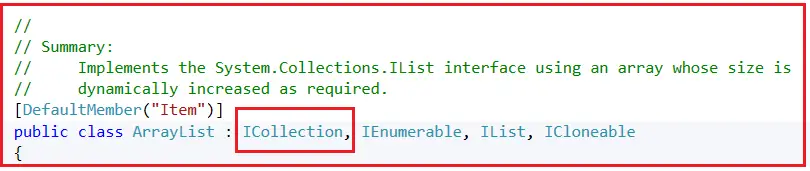
//Using var keyword without explicit casting

var firsItem = arrayList1**[**0**]**; //returns 101

var secondItem = arrayList1**[**1**]**; //returns "James"

//var fifthElement = arrayList1[5]; //Error: Index out of range

نکته : اگر به definition کلاس arrayList رفته باشید می بینید که این کلاس رابط ICollection را پیاده سازی کرده است این interface قابلیت iteration کردن بین عناصر آرایه را به ما می دهد به همین دلیل foreach را پشتیابنی می کند.



Add vs Insert

برای اضافه کردن یک ایتم به انتهای لیست می توان از Add استفاده کرد و همچنین برای افزودن یک ایتم به position به خصوصی می توان از insert استفاده کرد.

//Insert "First Element" at First Position i.e. Index 0

arrayList.Insert**(**0, "First Element"**)**;

InsertRange

برای افزودن مقادیر یک Collection به یک ArrayList به صورت جمعی می توان از InsertRange استفاده کرد دارای پارامتر index برای مشخص کردن این که از چه موقعیتی داده ها اضافه شوند به ArrayList

arrayList1.InsertRange**(**2, arrayList2**)**;

**How to Remove Elements from ArrayList in C#?**

1. **Remove(object? obj):** This method is used to remove the first occurrence of a specific object from the System.Collections.ArrayList. The parameter obj specifies the Object to remove from the ArrayList. The value can be null.
2. **RemoveAt(int index):** This method is used to remove the element at the specified index of the ArrayList. The parameter index specifies the index position of the element to remove.
3. **RemoveRange(int index, int count):** This method is used to remove a range of elements from the ArrayList. The parameter index specifies the starting index position of the range of elements to remove and the parameter count specifies the number of elements to remove.

##### **How to Remove all the elements from the ArrayList in C#?**

برای پاک کردن تمامی عناصر می توان از متد Clear استفاده کرد ضمن در نظر گرفتن این نکته که **این متد ظرفیت ArrayList را کاهش نمی دهد.**

**int** totalItems = arrayList.Count;

Console.WriteLine**(string**.Format**(**$"Total Items: {totalItems}, Capacity: {arrayList.Capacity}"**))**;

//Remove all items from the Array list

arrayList.Clear**()**;

totalItems = arrayList.Count;

Console.WriteLine**(string**.Format**(**$"Total Items After Clear(): {totalItems}, Capacity: {arrayList.Capacity}"**))**;

Console.ReadKey**()**;



**How do we Check whether an Element exists in ArrayList or not in C#?**

1. **bool Contains(object? item):** This method is used to determine whether an element is in the ArrayList. The parameter item specifies the Object to locate in the ArrayList. The value can be null. It returns true if the item is found in the ArrayList; otherwise, false.

**Note:**It is not recommended to use the non-generic collection class ArrayList in C# due to performance issues i.e. boxing and unboxing as it is operating on the object data type. So, instead of using ArrayList, it is recommended to use the generic collection List<object> to store heterogeneous objects. To store data of the same data type, use Generic List<T>.

**How to Clone the Non-Generic ArrayList Collection in C#?**

If you want to clone the Non-Generic ArrayList collection in C#, then you need to use the following Clone() method provided by the ArrayList Collection Class.

1. **Clone():** This method is used to create and return a shallow copy of the ArrayList.

**How to copy an ArrayList to an existing array in C#?**

1. **CopyTo(Array array):** This method is used to copy the entire ArrayList to a compatible one-dimensional Array, starting at the beginning of the target array. The parameter array specifies the one-dimensional Array that is the destination of the elements copied from ArrayList. The Array must have zero-based indexing. If the parameter array is null, then it will throw ArgumentNullException.
2. **CopyTo(Array array, int arrayIndex):** This method is used to copy the entire ArrayList to a compatible one-dimensional Array, starting at the specified index of the target array. Here, the parameter array specifies the one-dimensional array that is the destination of the elements copied from the ArrayList. The Array must have zero-based indexing. The arrayIndex parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter arrayIndex is less than zero, then it will throw ArgumentOutOfRangeException.
3. **CopyTo(int index, Array array, int arrayIndex, int count)**: This method is used to copy a range of elements from the System.Collections.ArrayList to a compatible one-dimensional Array, starting at the specified index of the target array. The index parameter specifies the zero-based index in the source System.Collections.ArrayList at which copying begins. The array parameter specifies the one-dimensional Array that is the destination of the elements copied from ArrayList. The Array must have zero-based indexing. The arrayIndex parameter specifies the zero-based index in the array at which copying begins. The count parameter specifies the number of elements to copy. If the parameter array is null, then it will throw ArgumentNullException. If the parameter index is less than zero, arrayIndex is less than zero, or the count is less than zero, then it will throw ArgumentOutOfRangeException.

**How to Sort the Elements of an ArrayList Collection in C#?**

If you want to sort the elements of the ArrayList in C#, then you can use the following Sort() method of the ArrayList class.

1. **Sort():** This method is used to sort the elements in the entire System.Collections.ArrayList.
2. **Sort(IComparer? comparer):** This method is used to sort the elements in the entire ArrayList using the specified comparer.
3. **Sort(int index, int count, IComparer? comparer):** This method is used to sort the elements in a range of elements in ArrayList using the specified comparer.

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

The ArrayList collection in C# is very much similar to the Arrays data type. The major difference between them is the dynamic nature of the non-generic collection ArrayList. For arrays, we need to define the size i.e. the number of elements that the array can hold at the time of array declaration. But in the case of the ArrayList collection in C#, this does not need to be done beforehand. Elements can be added or removed from the Array List collection at any point in time.

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

**Array:**

1. Fixed Length
2. Cannot insert it into the middle
3. Cannot delete from middle
4. It is type-safe, so we can store only similar types of data based on the data type.
5. Boxing and Unboxing are not required.

**ArrayList:**

1. Variable Length
2. Can insert an element into the middle of the collection
3. Can delete elements from the middle of the collection
4. It is not type-safe, so we can store any type of data.
5. Boxing and Unboxing are required as it is operated on the object data type.

# Hashtable in C#

از نوع non-generic است برای حل مشکلاتی است که با Array , ArraList داریم

**What are the Problems with Array and ArrayList Collection in C#?**

در Array , ArrayList دسترسی به عناصر با استفاده از Index است که از 0 تا تعداد کل – 1 متغییر است که برای ما سخت است که به ترتیب index تمامی اعضا را بدانیم.

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

al.Add**(**1001**)**; //EId - Index Position = 0

al.Add**(**"James"**)**; //Name - Index Position = 1

al.Add**(**"Manager"**)**; //Job - Index Position = 2

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

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

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

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

اگر تعداد زیاد باشد به خاطر سپردن index ها سخت است.

**راه حل استفاده از Key و Hashtable**

این نوع non generic collection برای ذخیره سازی داده ها در فرمت **key-value pair** کاربرد دارد.

**داده ها بر اساس کلید هش شده سازمان دهی می شوند.**

**نکته : کلید می تواند هر نوع داده ی باشد.**

این نوع برای هر کلید یک کد هش محاسبه می کند که برای جست جوی هر المان در مجموعه سبب افرایش سرعت می شود.

نکته : این collection رابط IDictionary را پیاده سازی کرده است

نکته: هر **کلید** باید unique باشد و null نباشد و می تواند هر نوع داده ی باشد.

نکته: مقادیر می تواندد تکراری و null باشد.

نکته: ظرفیت Hashtable بر اساس تعداد المان های که می تواند نگه دارد است.

نکته : به دلیل این که non-generic است می تواند هر نوع داده و نوع داده یکسانی را در خود نگه دارد.

**How Actually the Hashtable works in C#?**

زمانی که به Hashtable یک المان اضافه می کنیم از هر نوعی که باشد int string ,… سپس کلید مربوط به داده را فارغ ار نوع آن تبدیل به کد hash از نوع int تبدیل می کند به همین دلیل پیدا کردن آن سریع است زمانی که تبدیل کلید تمام شود سپس عملیات افزودن انجام می شود.

نکته : Hashtable performance is less than ArrayList because of key Conversion

**Differences between ArrayList and Hashtable in C#:**

1. **Lookup**: ArrayList can be only looked up via the index number which is generated internally. Hashtable can be looked up by a custom-defined key.
2. **Performance**: ArrayList is faster than hashtable because of extra tasks performed in hashtables i.e. hashing.
3. **Scenario**: If you want a key lookup use hashtable. If you just want to add and browser through a collection then use ArrayList.

**var cities = new Hashtable(){**  
**{“UK”, “London, Manchester, Birmingham”},**  
**{“USA”, “Chicago, New York, Washington”},**  
**{“India”, “Mumbai, Delhi, BBSR”}**  
**};**

**نکته : اگر کلید مربوطه در collection یافت نشد سبب بروز Exception می شود.**

**How to access**

با استفاده از حلقه بر روی keys یا با استفاده از DictionaryEntry به دلیل این که عناصر به صورت DictionaryEntry ذخیره می شوند.

**foreach (object obj in hashtable.Keys)**  
**{**  
**Console.WriteLine(obj + ” : ” + hashtable[obj]);**  
**}**

**foreach (DictionaryEntry item in hashtable)**  
**{**  
**Console.WriteLine($”Key: {item.Key}, Value: {item.Value}”);**  
**}**

**How to Check the Availability of a key/value Pair in a Hashtable in C#?**

بررسی وجود یا عدم وجود یک Key/value می توان از روش های زیر استفاده کرد.

1. **Contains(object key):** The Contains(object key) method of the Hashtable is used to check whether the Hashtable contains a specific key. The parameter key to locating in the Hashtable object. It returns true if the Hashtable contains an element with the specified key; otherwise, false. If the key is null, then it will throw System.ArgumentNullException.
2. **ContainsKey(object key):** The ContainsKey(object key) method of the Hashtable is used to check if a given key is present in the Hashtable or not. The parameter key to locating in the Hashtable object. If the given key is present in the collection then it will return true else it will return false. If the key is null, then it will throw System.ArgumentNullException.
3. **ContainsValue(object value):** The ContainsValue(object value) Method of the Hashtable class is used to check if a value is present in the Hashtable or not. The parameter value to locate in the hashtable object. If the given value is present in the collection then it will return true else it will return false.

**How to Remove Elements from a Non-Generic Hashtable Collection in C#?**

برای پاک کردن یک object بر اساس کلید آن می توانید از از متد زیر استفاده کنید.

1. **Remove(object key):** This method is used to remove the element with the specified key from the Hashtable. Here, the parameter key specifies the element to remove. It throws the KeyNotfoundException if the specified key is not found in the Hashtable, so check for an existing key using the ContainsKey() method before removing it.

برای پاک کردن تمامی عناصر می توان از متد **Clear** استفاده کرد.

**How to Assign Values to a Hashtable with Indexer in C#?**

برای اضافه کردن مقدار با استفاده از indexer می توان ابتدا کلید و سپس مقدار را مشخص کرد.

Hashtable hashtable = new Hashtable**()**;

hashtable**[**1**]** = "One";

hashtable**[**5**]** = "Five";

hashtable**[**30**]** = "Thirty";

**How to Update a Hashtable in C# using Indexer?**

Hashtable employee = new Hashtable

**{**

**{** "EId", 1001 **}**,

**{** "Name", "James" **}**,

**{** "Salary", 3500 **}**,

**{** "Location", "Mumbai" **}**,

**{** "EmailID", "a@a.com" **}**

**}**;

//Updating the Name and Salary

employee**[**"Name"**]** = "Smith"; //Update value of Name key

employee**[**"Salary"**]** = 5000; //Update value of Salary key

**How to Clone a Non-Generic Hashtable Collection in C#?**

با استفاده از متد Clone می توان یک کپی از collection ایجاد کرد و کپی آن را دریافت کرد.

//Creating a clone Hashtable using Clone method

Hashtable cloneHashtable = **(**Hashtable**)**hashtable.Clone**()**;

**How to Copy a Hashtable to an Existing Array in C#?**

برای کپی یک Hashtable به یک ارایه ی از قبل وجود دارد می توان از متد CopyTo() استفاده کرد

عملیات کپی با پیچیدگی زمان O(n) است که وابسته به تعداد زمان افزایش می یابد.

1. **CopyTo(Array array, int arrayIndex):** The CopyTo method of the Non-Generic Hashtable Collection Class is used to copy hashtable elements to a one-dimensional Array object, starting at the specified index in the array. Here, the parameter array specifies the one-dimensional Array object that is the destination of the DictionaryEntry objects copied from the hashtable. The Array must have zero-based indexing. The arrayIndex parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter arrayIndex is less than zero, then it will throw ArgumentOutOfRangeException.

نکته

1. To copy only the keys in the Hashtable, use Hashtable.Keys.CopyTo.
2. To copy only the values in the Hashtable, use Hashtable.Values.CopyTo.

مثال

//Copying the Hashtable to an object array

DictionaryEntry**[]** myArray = new DictionaryEntry**[**hashtable.Count**]**;

hashtable.CopyTo**(**myArray, 0**)**;

Console.WriteLine**(**"\nHashtable Copy Array Elements:"**)**;

**foreach** **(**DictionaryEntry item in myArray**)**

**{**

Console.WriteLine**(**$"Key: {item.Key}, Value: {item.Value}"**)**;

**}**

Object**[]** myObjArrayKey = new Object**[**hashtable.Count**]**;

Object**[]** myObjArrayValue = new Object**[**hashtable.Count**]**;

Console.WriteLine**(**"\nCopyTo Method to Copy Keys:"**)**;

hashtable.Keys.CopyTo**(**myObjArrayKey, 0**)**;

**foreach** **(**var key in myObjArrayKey**)**

**{**

Console.WriteLine**(**$"{key} "**)**;

**}**

Console.WriteLine**(**"\nCopyTo Method to Copy Values:"**)**;

hashtable.Values.CopyTo**(**myObjArrayValue, 0**)**;

**foreach** **(**var key in myObjArrayValue**)**

**{**

Console.WriteLine**(**$"{key} "**)**;

**}**

**Non-Generic Hashtable Collection Class Properties in C#**

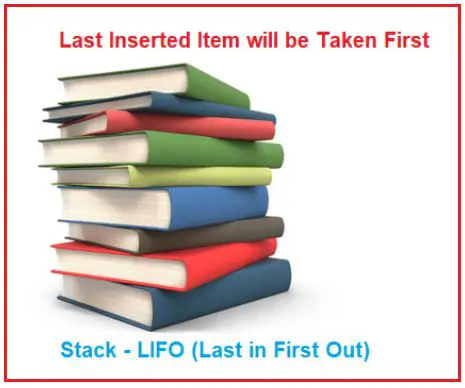
1. **IsFixedSize**: Gets a value indicating whether the System.Collections.Hashtable has a fixed size. It returns true if the Hashtable object has a fixed size; otherwise, false. The default is false.
2. **Keys**: Gets a System.Collections.ICollection containing the keys in the Hashtable. It returns a System.Collections.ICollection containing the keys in the Hashtable.
3. **IsSynchronized**: Gets a value indicating whether access to the Hashtable is synchronized (thread-safe). It returns true if access to the Hashtable is synchronized (thread-safe); otherwise, false. The default is false.
4. **IsReadOnly**: Gets a value indicating whether the Hashtable is read-only. It returns true if the Hashtable object is read-only; otherwise, false. The default is false.
5. **Count**: Gets the number of key/value pairs contained in the Hashtable. It returns the number of key/value pairs contained in the System.Collections.Hashtable.
6. **Values**: Gets a System.Collections.ICollection containing the values in the Hashtable. It returns a System.Collections.ICollection containing the values in the Hashtable.
7. **SyncRoot**: Gets an object that can be used to synchronize access to the Hashtable. It returns an object that can be used to synchronize access to the Hashtable.
8. **comparer**: Gets or sets the System.Collections.IComparer to use for the Hashtable. It returns the System.Collections.IComparer to use for the Hashtable.
9. **hcp**: Gets or sets the object that can dispense hash codes. It returns the object that can dispense hash codes.
10. **EqualityComparer**: Gets the System.Collections.IEqualityComparer to use for the Hashtable. It returns the System.Collections.IEqualityComparer to use for the Hashtable.

# **Non-Generic Stack Collection**

در این نوع از کالکشن روش Last in – First out استفاده می شود برای زمانی کاربرد دارد که ما نیاز به سیستم آخرین عضوی که وارد شده به عنوان اولین عضو خارج شود.

##### **What is Stack in C# and how does it work?**

یک کلاس غیر generic است. که به صورت LIFO کار می کند. اضافه کردن ایتم را psuh و خارج کردن ایتم از کالکشن را pop می گویند.



نکته : stack به صورت generic و هم به صورت non-generic پیاده سازی شده است.

##### **Methods, Properties, and Constructor of Stack Class in C#:**

برای هر دو حالت generic , non-generic در پیاده سازی هر دو رابط های IEnumerable. Icollection , ICloneable

Ctor

1. **Stack():** It is used to initialize a new instance of the Stack class that is empty and has the default initial capacity.
2. **Stack(ICollection col):** It is used to initialize a new instance of the non-generic Stack class that contains elements copied from the specified collection and has the same initial capacity as the number of elements copied. Here, the parameters col specifies the System.Collections.ICollection to copy elements from.
3. **Stack(int initialCapacity):** It is used to initialize a new instance of the System.Collections.Stack class that is empty and has the specified initial capacity or the default initial capacity, whichever is greater. Here, the parameter initialCapacity specifies the initial number of elements that the Stack can contain.

Stack stack = new Stack**()**;

//Adding item to the stack using the push method

stack.Push**(**10**)**;

stack.Push**(**"Hello"**)**;

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

stack.Push**(true)**;

stack.Push**(**67.8**)**;

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

//Printing the stack items using foreach loop

**foreach** **(object** item in stack**)**

**{**

Console.WriteLine**(**item**)**;

**}**

##### **How to Remove Elements from a Non-Generic Stack Collection in C#?**

1. **Pop():** This method is used to remove and return the object at the top of the Stack. It returns the Object (element) removed from the top of the Stack.
2. **Clear():** This method is used to remove all objects from the Stack.

##### **How to get the topmost element of a Stack in C#?**

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

##### **How to check whether an element exists or not in the stack in C#?**

1. **Contains(object obj):** This method is used to determine whether an element is in the Stack. Here, the parameter obj specifies the object or element to locate in the Stack. The value can be null. It returns true if obj is found in the Stack; otherwise, false.

نکته : مرتبه زمانی بررسی وجود یا عدم وجود یک object در stack دارای هزینه مرتبه زمانی O(n) است یعنی با توجه به تعداد زمان بیشتری می برد.

##### **How to Clone the Non-Generic Stack Collection in C#?**

1. **Clone():** This method is used to create and return a shallow copy of a stack object.

shallowCopy کپی که در آن یک object جدید ایجاد می شود و برای عناصر value-type آن ها را مستقیما کپی می کند و برای مقادیری که reference type هستند فقط reference را کپی می کند.

##### **How to copy a stack to an existing array in C#?**

1. **CopyTo(Array array, int index)**: This method is used to copy the Stack elements to an existing one-dimensional Array, starting at the specified array index. Here, the parameter array specifies the one-dimensional array that is the destination of the elements copied from the stack. The Array must have zero-based indexing. The index parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter index is less than zero, then it will throw ArgumentOutOfRangeException.
2. **object[]** stackCopy = new **object[**5**]**;
3. stack.CopyTo**(**stackCopy, 0**)**;

##### **Properties of Non-Generic Stack Collection Class in C#**

1. **Count**: It returns the number of elements contained in the Stack.
2. **IsSynchronized**: Gets a value indicating whether access to the Stack is synchronized (thread-safe). It returns true if access to the Stack is synchronized (thread-safe); otherwise, false. The default is false.
3. **SyncRoot**: Gets an object that can be used to synchronize access to the Stack. It returns an object that can be used to synchronize access to the Stack.

نکته : ظرفیت stack با افزودن عنصار اضافه می شود

نکته : اگر count از ظرفیت Stack کمتر باشد عملیات push دارای مرتبه زمانی O(1) بوده و اگر ظرفیت برای اضافه شدن عنصر موجود نباشد افزودن عنصر باعث افزایش ظرفیت شده و در این حالت عملیات دارای مرتبه زمانی O(n) می شود و عملیات pop در هر زمانی دارای مرتبه زمانی O(1) است.

# Non-Generic Queue in C#

ساختار عملکردی این نوع کالکشن به صورت first in – first out است به این معنی که اولین المانی که وارد کالکشن شود به عنوان اولین المان می تواند از صف خارج شود.

**What is a Queue and How Does the Queue Collection Work in C#?**

به صورت FIFO کار میکند به این معنی که اولین چیزی که اضافه می شود اولین چیزی است که حذف می شود.

عملیات افزودن enqueuing

عملیات حذف dequeuing



Mehtods

متد Enqueue برای افزودن عنصر به انتهای صف

متد Dequeue برای حذف قدیمی ترین عنصر صف

Peek برای دریافت قدیمی ترین عنصر بدون حذف از صف

ظرفیت برابر تعداد عناصر اضافه شده است و با افزودن عنصر ظرفیت به صورت خودکار اضافه می شود.

هم به صورت generic , non-generic است و امکان null و duplicate value نیز دارد.

Interfaces

Icollection – Ienumerable – Icloneable

**How to create a Queue in C#?**

1. **Queue():** It is used to initialize a new instance of the Queue class that is empty and has the default initial capacity, and uses the default growth factor.
2. **Queue(ICollection col):** It is used to initialize a new instance of the Queue class that contains elements copied from the specified collection and has the same initial capacity as the number of elements copied and uses the default growth factor. Here, the parameters col specifies the System.Collections.ICollection to copy elements from.
3. **Queue(int capacity):** It is used to initialize a new instance of the Queue class that is empty, has the specified initial capacity, and uses the default growth factor. Here, the parameter capacity specifies the initial number of elements that the Queue can contain.
4. **Queue(int capacity, float growFactor):** It is used to initialize a new instance of the Queue class that is empty, has the specified initial capacity, and uses the specified growth factor. Here, the parameter capacity specifies the initial number of elements that the Queue can contain and the parameter growFactor specifies the factor by which the capacity of the Queue is expanded.

**growFactor**

**مشخص کننده میزان رشد ظرفیت زمانی که به فضای بیشتری نیاز داریم. مشخص می کند با چه سرعتی ظرفیت رشد کند.**

* Default value is 2.0 (meaning capacity doubles when needed)
* Must be between 1.0 and 10.0
* Minimum capacity increase is always 4 slots, regardless of the growth factor

**How to Remove Elements from the Queue Collection in C#?**

1. **Dequeue():** This method is used to remove and return the object at the beginning of the Queue. It returns the object that is removed from the beginning of the Queue.
2. **Clear():** This method is used to remove all objects from the queue.

How to get first element of the queue

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

##### **How to Check Whether an Element Exists or not in the Queue Collection in C#?**

1. **Contains(object obj):** This method is used to determine whether an element is in the Queue. Here, the parameter obj specifies the object or element to locate in the Queue. The value can be null. It returns true if obj is found in the queue; otherwise, false.

**How to Clone the Non-Generic Queue Collection in C#?**

1. **Clone():** This method is used to create and return a shallow copy of a Queue object.

Queue cloneQueue = **(**Queue**)**queue.Clone**()**;

**How to copy a queue to an existing array in C#?**

1. **CopyTo(Array array, int index):** The CopyTo method of the Non-Generic Queue Collection Class in C# is used to copy the System.Collections.Queue elements to an existing one-dimensional System.Array, starting at the specified array index. Here, the parameter array specifies the one-dimensional Array that is the destination of the elements copied from Queue. The Array must have zero-based indexing. The index parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter index is less than zero, then it will throw ArgumentOutOfRangeException.
2. **object[]** queueCopy = new **object[**5**]**;
3. queue.CopyTo**(**queueCopy, 0**)**;

# Non-Generic SortedList in C#

از نوع non-generic است و به صورت key/value می باشد یک لیست مرتب که با index و key قابل دسترسی است.

به صورت پیش فرض مرتب سازی به صورت **ascending یا صعودی است.**

**Properties of Non-Generic SortedList Class in C#**

رابط های IEnumerable , ICollection , IDictionary , ICloneable را پیاده سازی

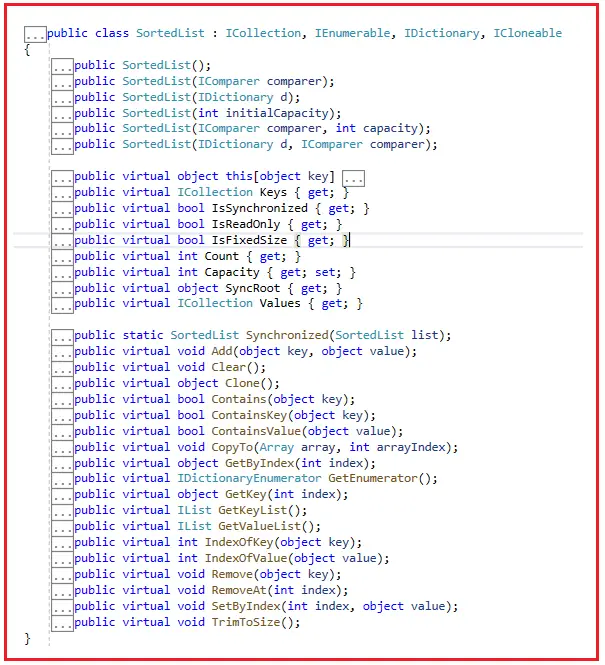
قابلیت دسترسی به عنصر با index و کلید

از 2 آرایه برای ذخیره سازی داده ها استفاده می کند یک آرایه برای نگه داری کلید و دیگری و دیگری برای نگه داری مقدار

نکته : کلید نمی تواند دارای مقدار null باشد.

ظرفیت بر اساس تعداد عناصری است که نگه می دارد. می توان مقادیر یک نوع یا چندین نوع را در خود نگه داری کند

نکته : کلید نمی تواند از نوع مختلف باشد.



1. **SortedList():** It initializes a new instance of the System.Collections.SortedList class that is empty, has the default initial capacity and is sorted according to the IComparable interface implemented by each key added to the System.Collections.SortedList object.
2. **SortedList(IComparer comparer):** It initializes a new instance of the System.Collections.SortedList class that is empty, has the default initial capacity and is sorted according to the specified IComparer interface. The Parameter comparer specifies the System.Collections.IComparer implementation to use when comparing keys. -or- null to use the System.IComparable implementation of each key.
3. **SortedList(IDictionary d):** IT initializes a new instance of the System.Collections.SortedList class contains elements copied from the specified dictionary, has the same initial capacity as the number of elements copied, and is sorted according to the System.IComparable interface implemented by each key. The Parameter d specifies the System.Collections.IDictionary implementation to copy to a new System.Collections.SortedList object.
4. **SortedList(int initialCapacity):** It initializes a new instance of the System.Collections.SortedList class that is empty, has the specified initial capacity, and is sorted according to the System.IComparable interface implemented by each key added to the System.Collections.SortedList object. The Parameter initialCapacity specifies the initial number of elements that the System.Collections.SortedList object can contain.
5. **SortedList(IComparer comparer, int capacity):** It initializes a new instance of the System.Collections.SortedList class that is empty, has the specified initial capacity, and is sorted according to the specified System.Collections.IComparer interface. The Parameter comparer specifies the System.Collections.IComparer implementation to use when comparing keys. -or- null to use the System.IComparable implementation of each key. The Parameter capacity specifies the initial number of elements that the System.Collections.SortedList object can contain.
6. **SortedList(IDictionary d, IComparer comparer):** It initializes a new instance of the System.Collections.SortedList class contains elements copied from the specified dictionary, has the same initial capacity as the number of elements copied, and is sorted according to the specified System.Collections.IComparer interface. The Parameter d specifies the System.Collections.IDictionary implementation to copy to a new System.Collections.SortedList object. The Parameter comparer specifies the System.Collections.IComparer implementation to use when comparing keys. -or- null to use the System.IComparable implementation of each key.

**How to Add Elements into a SortedList in C#?**  
**SortedList sortedList = new SortedList();**  
**sortedList.Add(1, “One”);**  
**sortedList.Add(3, “Three”);**

You can also store a key/value pair in the SortedList using Collection Initializer as follows.  
**SortedList sortedList = new SortedList**  
**{**  
**{ 1, “One” },**  
**{ 3, “Three” }**  
**};**

**How to Remove Elements from a SortedList Collection in C#?**

1. **Remove(object key):** This method is used to remove the element with the specified key from a System.Collections.SortedList object. The parameter key specifies the element to remove.
2. **RemoveAt(int index):** This method is used to remove the element at the specified index of a System.Collections.SortedList object. The parameter index specifies the element to remove. It is 0 based Index.
3. **Clear()**: This method is used to remove all elements from a System.Collections.SortedList object.

**How to check the availability of key/value pairs in a SortedList in C#?**

1. **Contains(object key):** This method is used to determine whether the SortedList object contains a specific key. The parameter key to locate in the SortedList object. It returns true if the SortedList object contains an element with the specified key; otherwise, false. If the key is null, then it will throw System.ArgumentNullException.
2. **ContainsKey(object key):** This method is used to determine whether a SortedList object contains a specific key. The parameter key to locate in the SortedList object. It returns true if the SortedList object contains an element with the specified key; otherwise, false. If the key is null, then it will throw System.ArgumentNullException.
3. **ContainsValue(object value):** This method is used to determine whether a System.Collections.SortedList object contains a specific value. The parameter value to locate in the SortedList object. The value can be null. It returns true if the SortedList object contains an element with the specified value; otherwise, false.

##### **How to Clone the Non-Generic SortedList in C#?**

1. **Clone():** This method is used to create and return a shallow copy of a SortedList object.

Example

SortedList cloneSortedList = **(**SortedList**)**sortedList.Clone**()**;

**CopyTo(Array array, int arrayIndex):** The CopyTo method of the Non-Generic SortedList Collection Class in C# is used to copy SortedList elements to a one-dimensional Array object, starting at the specified index in the array. Here, the parameter array specifies the one-dimensional Array object that is the destination of the DictionaryEntry objects copied from SortedList. The Array must have zero-based indexing. The arrayIndex parameter specifies the zero-based index in the array at which copying begins. If the parameter array is null, then it will throw ArgumentNullException. If the parameter arrayIndex is less than zero, then it will throw ArgumentOutOfRangeException.

**کپی داده ها به همان ترتیبی که ذخیره شده اند اتفاق می افتد و پیچیدگی زمانی برابر است با O(n) که n برابر Count است**

**امکان کپی کردن صرفا key یا value نیز وجود دارد.**

1. **To copy only the keys in the SortedList, use SortedList.Keys.CopyTo.**
2. **To copy only the values in the SortedList, use SortedList.Values.CopyTo.**

##### **When to Use Non-Generic SortedList Collection in C#?**

برای مواقعی که نیاز به تغییر سریع داده ها بر اساس key/value به ترتیب منظم کاربرد دارد.

**نکته: به دلیل این که با هر بار اضافه کردن مقدار به کالکشن یا حذف مقدار بر اساس ماهیت این نوع لازم است که همیشه مرتب باشد پس عملیات مرتب سازی انجام می پذیرد. که با افزایش تعداد عناصر cost بالا می رود.**

باید زمانی از SortedList استفاده کنیم که کالکشن کوچکی نیاز داریم که باید تمام مدت مرتب باشد

مواقعی که دیتا زیاد است باید از مواردی مانند Dictionay , HastSet , List استفاده کنیم

# Advantages and Disadvantages of Non-Generic Collection in C#

* کلاس های non-generic افزایش ظرفیت خودکار دارند زمانی که به آن ها ایتم اضافه می شود و این از مزایای آن ها است. ArrayList, Stack, Queue, SortedList, and Hashtable,…

**مثال**

آرایه ی از نوع non-generic با ظرفیت 3 ایجاد می کنیم سپس 4 ایتم به آن اضافه می کنیم.

ArrayList Numbers = new ArrayList**(**3**)**;

Numbers.Add**(**100**)**;

Numbers.Add**(**200**)**;

Numbers.Add**(**300**)**;

Numbers.Add**(**400**)**;

بدون این که خطای run time , compile time داشته باشیم.

* امکان اضافه یا حذف ایتم از میانه ی کالکشن که در آرایه ها امکان پذیر نیست اما در collection امکان پذیره.

معایب

به دلیل این که loosely type هستند امکان برخورد با خطای runtime وجود دارد. Loosely type به معنی استفاده از نوع object است است به عنوان value .

نه تنها به دلیل این که loosely type سبب exception می شود بلکه به دلیل این که عملیات boxing , unboxing برای object رخ می دهد بر روی performance تاثیر گذار است.

یعنی مقادیر 100 و 200 را به کالکشن می دهیم از نوع int هستند و این نوع نیز value type هست اما نوع object از نوع از نوع reference type است پس بنابراین باید مقادیر به reference type تبدیل شوند و عملیات boxing رخ دهد.

همچنین برای retrieve کردن عناصر لازم است که از نوع object تبدیل به نوع int رخ دهند که عملیات unboxing است.

پس بنابراین باید از کالشکن های بزرگ از این نوع پرهیز شود چون عملیات boxing , unboxing یک عمل هزینه بر است.

# Generic Collections in C#

Generic collection ها برای از بین بردن معایب مربوط به non-generic ها کاربرد دارد در این نوع collection ها باید نوع داده ای که می خواهیم ذخیره کنیم را مشخص کنیم.

Strongly type safe بودن از مزایای این نوع از کالکشن ها است. و فقط یک نوع داده را می توان به کالکشن اضافه کرد.

1. [**List<T>**](https://dotnettutorials.net/lesson/list-collection-csharp/)**:** It represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists. It grows automatically as you add elements to it.
2. [**Dictionary<TKey, Tvalue>**](https://dotnettutorials.net/lesson/dictionary-generic-collection-csharp/)**:** It represents a collection of keys and values.
3. [**SortedSet<T>:**](https://dotnettutorials.net/lesson/generic-sortedset-collection-class-in-csharp/)It represents a collection of objects that are maintained in sorted order.
4. [**SortedDictionary<TKey, TValue>**](https://dotnettutorials.net/lesson/generic-sorteddictionary-collection-class-in-csharp/)**:** It represents a collection of key/value pairs that are sorted on the key.
5. [**LinkedList<T>**](https://dotnettutorials.net/lesson/generic-linkedlist-collection-class-in-csharp/)**:** It represents a doubly linked list.

از مزایای آن ها می توان به عملکرد مناسب به دلیل عدم نیاز به boxing , unboxing می توان اشاره کرد. و همچنین از بروز خطا های run time به دلیل نوع داده جلوگیری می کند.

**List<T> Class in C#**

می تواند مقدار تکراری داشته باشد

**HashSet<T> Class in C#**

فقط مقادیر unique می گیرد و امکان ذخیره سازی مقادیر تکراری را ندارد. می توانید مقدار تکراری اضافه کنید اما در نهایت خود کالکشن مقدار تکراری را حذف می کند

**SortedSet<T> Class in C#:**

عدم امکان ذخیره سازی مقادیر تکراری و مناسب نگه داری مقادیر unique

**SortedDictionary<TKey, TValue> Class in C#**

کلید باید غیر تکراری باشد. و ترتیب ascending یا صعودی است.

**LinkedList<T> Class in C#:**

اجازه ی افزودن و حذف عنصر با سرعت بالا را دارد. می تواند مقادیر تکراری نیز داشته باشد.

این امکان را به ما می دهد که قبل یا بعد از آخرین index عناصری را اضافه کنیم.

قابلیت AddFirst , AddLast را نیز دارد

linkedList.AddLast**(**"Three"**)**;

linkedList.AddLast**(**"Four"**)**;

linkedList.AddFirst**(**"Five"**)**;

# **Generic Constraints in C#**

Constraint به این معنا است که می توان محدودیت های برای محدود کردن type که می تواند استفاده شود کاربرد دارد.

به شما خطای compile-time می دهد در صورتی که بخواهید نوعی غیر از نوع مشخص شده توسط constraint به generic الحاق کنید.

می توانید یک یا چند constraint را بر روی یک generic با استفاده از کلمه کلیدی where اعمال کنید.

اگر می خواهید که generic شما فقط یک نوع خاص را دریافت کند باید از constraint ها استفاده کنید و نوع قابل قبول را مشخص کنید. و اگر نوع درست را استفاده نکنید باعث خطای compile time می شود.

##### **Types of Generic Constraints in C#**

1. **where T: struct** => The type argument must be non-nullable value types such as primitive data types int, double, char, bool, float, etc. The struct constraint can’t be combined with the unmanaged constraint.
2. **where T: class** => The type argument must be a reference type. This constraint can be applied to any class (non-nullable), interface, delegate, or array type in C#.
3. **where T: new()** => The type argument must be a reference type that has a public parameterless (default) constructor.
4. **where T: <base class name>** => The type of argument must be or derive from the specified base class.
5. **where T: <interface name>** => The type argument must be or implement the specified interface. Also, multiple interface constraints can be specified.
6. **where T: U** => The type argument supplied for must be or derive from the argument supplied for U. In a nullable context, if U is a non-nullable reference type, T must be a non-nullable reference type. If U is a nullable reference type, T may be either nullable or non-nullable.

T:Class به این معنی است که فقط مقادیر reference type قابل پذیرش هستند .

 In C#, the string is a reference type.  
**GenericClass<string> stringClass = new GenericClass<string>();**

**X**

The following statement will give you a compile-time error as int is a value type, not a reference type.  
**GenericClass<int> intClass = new GenericClass<int>();**

Struct به این معنی است که مقادیر Value type قابل پذیرش هستند.

create an instance of GenericClass by passing value-type arguments as follows.  
**GenericClass<int> intClass = new GenericClass<int>();**

**X**

The following statement will give you a compile-time error as the string is a reference type, not a value type.  
**GenericClass<string> stringClass = new GenericClass<string>();**

#### **where T: BaseClass Generic Constraint in C#**

به این معنی است که می توان یک کلاس مشخص را به عنوان constraint مشخص کرد و تمامی کلاس های که از آن کلاس مشخص ارث برده اند نیز آن استفاده کنند

**public** **class** GenericClass**<**T**>** **where** T : BaseEmployee

**{**

**public** T Message;

**public** **void** GenericMethod**(**T Param1, T Param2**)**

**{**

Console.WriteLine**(**$"Message: {Message}"**)**;

Console.WriteLine**(**$"Param1: {Param1}"**)**;

Console.WriteLine**(**$"Param2: {Param2}"**)**;

**}**

**}**

**public** **class** BaseEmployee

**{**

**}**

**public** **class** Employee : BaseEmployee

**{**

**public** **string** Name **{** **get**; **set**; **}**

**}**

**public** **class** Customer

**{**

**public** **string** Name **{** **get**; **set**; **}**

**}**

 It works fine because Employee is derived from the BaseEmployee class.  
**GenericClass<Employee> employee = new GenericClass<Employee>();**

The following statement will give you a compile-time error as the Customer class is not derived from the BaseEmployee type.  
**GenericClass<Customer> customer = new GenericClass<Customer>();**

Interface constraint

می توان به عنوان محدودیت نوع interface را نیز مشخص کرد که در این حالت فقط کلاسی می تواند در generic باشد که فرآیند ساخت نیز مشارکت داشته اند.

**public** **class** GenericClass**<**T**>** **where** T : IEmployee

**{**

**public** T Message;

**public** **void** GenericMethod**(**T Param1, T Param2**)**

**{**

Console.WriteLine**(**$"Message: {Message}"**)**;

Console.WriteLine**(**$"Param1: {Param1}"**)**;

Console.WriteLine**(**$"Param2: {Param2}"**)**;

**}**

**}**

let us create an instance of GenericClass bypassing Employee type arguments as follows. It works fine because the Employee class implements the IEmployee interface.  
**GenericClass<Employee> employee = new GenericClass<Employee>();**

The following statement will give you a compile-time error as the Customer class does not implement the IEmployee interface.  
**GenericClass<Customer> customer = new GenericClass<Customer>();**

#### **where T: U Generic Constraint in C#**

Here, the type argument supplied must be or derive from the argument supplied for U. In a nullable context, if U is a non-nullable reference type, T must be a non-nullable reference type. If U is a nullable reference type, T may be either nullable or non-nullable. So, in this constraint, there are two Type Arguments i.e. T and U. U can be an interface, abstract class, or simple class. T must inherit or implements the U class. For a better understanding, please have a look at the below code.

**using** *System;*

**namespace** *GenericsDemo*

**{**

**public** **class** GenericClass**<**T, U**>** **where** T : U

**{**

**public** T Message;

**public** **void** GenericMethod**(**T Param1, U Param2**)**

**{**

Console.WriteLine**(**$"Message: {Message}"**)**;

Console.WriteLine**(**$"Param1: {Param1}"**)**;

Console.WriteLine**(**$"Param2: {Param2}"**)**;

**}**

**}**

**}**

As you can see in the above code, here we have used **where T: U** constraint which allows the type (T) that must inherit or implements the U class. Now, let us create one interface and two more classes as follows.

**namespace** *GenericsDemo*

**{**

**public** **interface** IEmployee

**{**

**}**

**public** **class** Employee : IEmployee

**{**

**public** **string** Name **{** **get**; **set**; **}**

**}**

**public** **class** Customer

**{**

**public** **string** Name **{** **get**; **set**; **}**

**}**

**}**

As you can see in the above code, the Employee class implements the IEmployee interface. On the other hand, the Customer class is not implementing the IEmployee interface. Now, Let’s create an instance of Genericclass bypassing Employee and IEmployee as type arguments for T and U as follows. It works fine because the Employee class implements the IEmployee interface.

**GenericClass<Employee, IEmployee> employeeGenericClass = new GenericClass<Employee, IEmployee>();**

The following statement will give you a compile-time error as the Customer class does not implement the IEmployee interface i.e. T does not implement U.  
**GenericClass<Customer, IEmployee> customerGenericClass = new GenericClass<Customer, IEmployee>();**

#### **Multiple Generic Constraints in C#:**

In C# generics, it is also possible to apply multiple constraints on generic classes based on our requirements. Let us understand this with an example. In the below example, we are creating the generic class with two constraints. The first constraint specifies that the T parameter must be a reference type whereas the second constraint specifies that the X parameter must be a value type.

**public** **class** GenericClass**<**T, X**>** **where** T: **class** **where** X: struct

**{**

**public** T Message;

**public** **void** GenericMethod**(**T Param1, X Param2**)**

**{**

Console.WriteLine**(**$"Message: {Message}"**)**;

Console.WriteLine**(**$"Param1: {Param1}"**)**;

Console.WriteLine**(**$"Param2: {Param2}"**)**;

**}**

**}**

**class** Program

**{**

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

**{**

GenericClass**<string**, **int>** multipleGenericConstraints = new GenericClass**<string**, **int>()**;

multipleGenericConstraints.Message = "Good Morning";

multipleGenericConstraints.GenericMethod**(**"Anurag", 100**)**;

Console.ReadKey**()**;

**}**

**}**