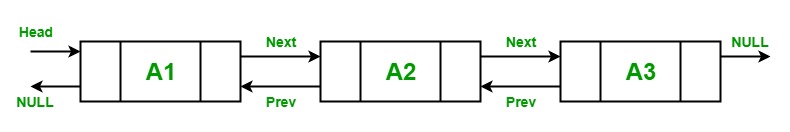
C# Collections

# C# LinkedList<T>

1. A Linear data structure which stores element in the non-contiguous location.
2. LinkedList is the generic type of collection which is defined in System.Collections.Generic namespace.
3. The elements in a linked list are linked with each other using pointers. (Nodes where each node contains a data field and a reference (link) to the next node in the list).
4. Provides fast inserting and removing elements.



1. A node in the LinkedList<T> object is of the type LinkedListNode<T>.
2. The First and Last Properties of an Empty Linked List is NULL.
3. The capacity of a LinkedList is the number of elements the LinkedList can hold.
4. The LinkedList class implements the ICollection<T>, IEnumerable<T>, IReadOnlyCollection<T>, ICollection, IEnumerable, IDeserializationCallback, and ISerializable interfaces.
5. Nodes can be Added, Removed anywhere in the middle of the list. This results in no additional objects allocated on the heap.
6. Linked Lists does not support chaining, splitting, cycles, or other features that can leave the list in an inconsistent state.

## Constructors

**LinkedList():**

This constructor is used to create an instance of the LinkedList class that is empty.

**LinkedList(IEnumerable):**

This constructor is used to create an instance of the LinkedList class that contains elements copied from the specified IEnumerable and has sufficient capacity to accommodate the number of elements copied.

**LinkedList(SerializationInfo, StreamingContext):**

This constructor is used to create an instance of the LinkedList class that is serializable with the specified SerializationInfo and StreamingContext.

## Methods

**AddAfter**: This method is used to add a new node or value after an existing node in the LinkedList.

**AddBefore**: This method is used to add a new node or value before an existing node in the LinkedList.

**AddFirst**: This method is used to add a new node or value at the start of the LinkedList.

**AddLast**: This method is used to add a new node or value at the end of the LinkedList.

**Clear():** This method is used to remove all nodes from the LinkedList.

**Remove(LinkedListNode):** This method is used to remove the specified node from the LinkedList.

**Remove(T):** This method is used to remove the first occurrence of the specified value from the LinkedList.

**RemoveFirst():** This method is used to remove the node at the start of the LinkedList.

**RemoveLast():** This method is used to remove the node at the end of the LinkedList.

**Contains(T):** In LinkedList, you can check whether the given value is present or not using the.

# C# Sorted Lists

The SortedList<TKey, TValue>, and SortedList are collection classes that can store key-value pairs that are sorted by the keys based on the associated IComparer implementation. If the keys are of primitive types, then they are sorted in ascending order of keys.

1. **Namespace**: System.Collection.Generic.
2. SortedList<TKey, TValue> is an array of key-value pairs sorted by keys.
3. Sorts elements as soon as they are added. Sorts primitive type keys in ascending order and object keys based on IComparer<T>.
4. A key must be unique and cannot be null.
5. A value can be null or duplicate.
6. A value can be accessed by passing associated key in the indexer mySortedList[key]
7. Contains elements of type KeyValuePair<TKey, TValue>
8. It uses less memory than SortedDictionary<TKey,TValue>.
9. It is faster in the retrieval of data once sorted, whereas SortedDictionary<TKey, TValue> is faster in insertion and removing key-value pairs.
10. The Keys cannot be duplicated or be NULL.

# C# Sets

A **HashSet<T>** is an unordered collection of the unique elements. It comes under **System.Collections.Generic** namespace. It is used in a situation where we want to prevent duplicates from being inserted in the collection. As far as performance is concerned, it is **better in comparison to the list**.

1. The HashSet<T> class provides high-performance set operations. A set is a collection that contains no duplicate elements, and whose elements are in no particular order.
2. The capacity of a HashSet<T> object is the number of elements that the object can hold.
3. A HashSet<T> object’s capacity automatically increases as elements are added to the object.
4. A HashSet<T> collection is not sorted and cannot contain duplicate elements.
5. HashSet<T> provides many mathematical set operations, such as set addition (unions) and set subtraction.