```
using System;
using System.Collections.Generic;

namespace _1.LIST
{
    #region Definition of Class Players
    /// <summary>
    /// Players is a custom class (complex type) with multiple properties.
    /// We'll use this to demonstrate storing objects in a generic List<T>.
    /// </summary>
    public class Players
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Country { get; set; }
        public char Grade { get; set; }
        public int IccRank { get; set; }
    }
    #endregion

    class Program
    {
        static void Main(string[] args)
        {
            #region What is Generic Type?
            /*
             * A "generic type" means we can create a type (like List<T>)
             * where 'T' can be replaced with any data type (int, string, class, etc.).
             * It provides type-safety and reusability.
             */
            #endregion

            #region What is List<T>?
            /*
             * List<T> is a generic collection class in C#.
             * It is used to store multiple elements of the same type dynamically.
             * It grows/shrinks automatically (unlike arrays which are fixed size).
             */
            #endregion

            #region Example: List with Simple Type (string)
            List<string> players = new List<string>();

            // Initialize with collection
            List<string> indianPlayers = new List<string>
            {
                "M.S. Dhoni", "Sachin", "Bumrah"
            };

            // Add elements
            players.Add("Rohit Sharma");
            players.Add("Virat Kohli");
            players.Add("Pollard");

            // Add multiple items
            players.AddRange(indianPlayers);

            // Insert at specific index
            players.Insert(0, "Shami");
            players.InsertRange(0, indianPlayers);

            // Check existence
```

```
            Console.WriteLine("Is 'Virat Kohli' in IndianPlayers? " +
indianPlayers.Contains("Virat Kohli"));

            // Remove elements
            indianPlayers.Remove("Sachin");            // value based
            players.RemoveAt(0);                       // index based
            players.RemoveRange(0, 2);                 // index + count
            players.RemoveAll(x => x.Length > 10);     // predicate (condition)

            // Clear list
            players.Clear();

            // Count property
            Console.WriteLine("Total Indian Players: " + indianPlayers.Count);

            // Sort and reverse
            indianPlayers.Sort();    // ascending
            indianPlayers.Reverse(); // descending

            // Update
            indianPlayers[0] = "Suryakumar Yadav";

            // Access elements (index based)
            for (int i = 0; i < indianPlayers.Count; i++)
            {
                Console.WriteLine($"indianPlayers[{i}] = {indianPlayers[i]}");
            }

            // Access elements (foreach)
            foreach (string player in indianPlayers)
            {
                Console.WriteLine(player);
            }
            #endregion

            #region Example: List with Complex Type (Players class)
            // Create objects of Players class
            Players p1 = new Players { Id = 1, Name = "Rohit Sharma", Country = "India", Grade =
'A', IccRank = 1 };
            Players p2 = new Players { Id = 2, Name = "Virat Kohli", Country = "India", Grade =
'A', IccRank = 2 };
            Players p3 = new Players { Id = 3, Name = "Bumrah", Country = "India", Grade = 'A',
IccRank = 3 };
            Players p4 = new Players { Id = 4, Name = "Suryakumar Yadav", Country = "India", Grade
= 'B', IccRank = 4 };
            Players p5 = new Players { Id = 5, Name = "Maxwell", Country = "Australia", Grade =
'C', IccRank = 10 };
            Players p6 = new Players { Id = 6, Name = "Brevis", Country = "South Africa", Grade =
'A', IccRank = 1 };

            // Create list of complex type
            List<Players> playerList = new List<Players> { p1, p2, p3, p4, p5, p6 };

            // Find examples
            var firstIndian = playerList.Find(x => x.Country == "India"); // first match
            Console.WriteLine("First Indian Player: " + firstIndian.Name);

            var allIndians = playerList.FindAll(x => x.Country == "India"); // all matches
            Console.WriteLine("All Indian Players:");
            foreach (var player in allIndians)
            {
                Console.WriteLine(player.Name);
```

```
            }

            int indexOfBumrah = playerList.FindIndex(x => x.Name == "Bumrah");
            Console.WriteLine("Index of Bumrah: " + indexOfBumrah);

            var lastGradeA = playerList.FindLast(x => x.Grade == 'A');
            Console.WriteLine("Last Grade A Player: " + lastGradeA.Name);

            int lastIndexIndia = playerList.FindLastIndex(x => x.Country == "India");
            Console.WriteLine("Last Index of Indian Player: " + lastIndexIndia);

            bool existsMaxwell = playerList.Exists(x => x.Name == "Maxwell");
            Console.WriteLine("Does Maxwell exist? " + existsMaxwell);
            #endregion
        }
    }
}
```

```
using System;
using System.Collections.Generic;


    #region Definition of Class Players
    /// <summary>
    /// Players is a custom class (complex type) with multiple properties.
    /// We'll use this to demonstrate storing objects in a generic List<T>.
    ///
    /// Implements IComparable<Players> so we can define a default sorting rule.
    /// </summary>
    public class Players : IComparable<Players>
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Country { get; set; }
        public char Grade { get; set; }
        public int IccRank { get; set; }

        // Default sorting rule → By ICC Rank (ascending)
        public int CompareTo(Players other)
        {
            if (other == null) return 1;
            return this.IccRank.CompareTo(other.IccRank);
        }

        public override string ToString()
        {
            return $"{Name} ({Country}) - Rank {IccRank}, Grade {Grade}";
        }
    }
    #endregion

    #region Custom Comparers
    /// <summary>
    /// Sort players by Name (Alphabetical)
    /// </summary>
    public class SortByName : IComparer<Players>
    {
        public int Compare(Players x, Players y)
        {
            return string.Compare(x?.Name, y?.Name, StringComparison.OrdinalIgnoreCase);
        }
    }
```

```
    /// <summary>
    /// Sort players by Country
    /// </summary>
    public class SortByCountry : IComparer<Players>
    {
        public int Compare(Players x, Players y)
        {
            return string.Compare(x?.Country, y?.Country, StringComparison.OrdinalIgnoreCase);
        }
    }
    #endregion
    class Program
    {
        static void Main(string[] args)
        {
            #region Create List of Complex Type
            List<Players> playerList = new List<Players>
            {
                new Players { Id = 1, Name = "Rohit Sharma", Country = "India", Grade = 'A',
IccRank = 1 },
                new Players { Id = 2, Name = "Virat Kohli", Country = "India", Grade = 'A',
IccRank = 2 },
                new Players { Id = 3, Name = "Bumrah", Country = "India", Grade = 'A', IccRank =
3 },
                new Players { Id = 4, Name = "Suryakumar Yadav", Country = "India", Grade = 'B',
IccRank = 4 },
                new Players { Id = 5, Name = "Maxwell", Country = "Australia", Grade = 'C',
IccRank = 10 },
                new Players { Id = 6, Name = "Brevis", Country = "South Africa", Grade = 'A',
IccRank = 1 }
            };
            #endregion

            #region Default Sorting (By ICC Rank using IComparable)
            Console.WriteLine("---- Sort by ICC Rank (Default CompareTo) ----");
            playerList.Sort();
            foreach (var player in playerList)
                Console.WriteLine(player);
            #endregion

            #region Custom Sorting (By Name using IComparer)
            Console.WriteLine("\n---- Sort by Name ----");
            playerList.Sort(new SortByName());
            foreach (var player in playerList)
                Console.WriteLine(player);
            #endregion

            #region Custom Sorting (By Country using IComparer)
            Console.WriteLine("\n---- Sort by Country ----");
            playerList.Sort(new SortByCountry());
            foreach (var player in playerList)
                Console.WriteLine(player);
            #endregion

            #region Inline Sorting with Lambda Expression
            Console.WriteLine("\n---- Sort by Grade (using Lambda) ----");
            playerList.Sort((x, y) => x.Grade.CompareTo(y.Grade));
            foreach (var player in playerList)
                Console.WriteLine(player);
            #endregion
        }
```

```
    }
```

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace _2.Dictionary
{
    class Program
    {
        static void Main(string[] args)
        {
            #region What is Dictionary<TKey, TValue>?
            /*
             * Dictionary<TKey, TValue> is a generic collection that stores data as key-value
pairs.
             * - Key: must be unique.
             * - Value: can be duplicate.
             * - Provides fast lookup (search by key).
             */
            #endregion

            #region Create a Dictionary
            Dictionary<int, string> products = new Dictionary<int, string>();
            #endregion

            #region Add Elements
            products.Add(1, "OnePlus");
            products.Add(5, "Samsung");
            products.Add(2, "Redmi");
            products.Add(4, "Nothing");
            #endregion

            #region Update Element
            // Update value for a given key
            products[2] = "Apple"; // Key=2 updated from Redmi → Apple
            #endregion

            #region Access Elements
            // Access using for loop (via Keys collection)
            for (int i = 0; i < products.Count; i++)
            {
                var key = products.Keys.ElementAt(i);
                var value = products[key];
                Console.WriteLine($"Key={key}, Value={value}");
            }

            // Access using foreach (recommended)
            foreach (KeyValuePair<int, string> kvp in products)
            {
                Console.WriteLine($"Key={kvp.Key}, Value={kvp.Value}");
            }
            #endregion

            #region Check Existence
            Console.WriteLine("Contains Key 1? " + products.ContainsKey(1)); // True
            Console.WriteLine("Contains Value 'OnePlus'? " + products.ContainsValue("OnePlus"));
// True
            #endregion

            #region Remove Elements
```

```
            bool removed = products.Remove(4); // Remove key=4, returns true if successful
            Console.WriteLine("Was key 4 removed? " + removed);

            // Clear all elements
            products.Clear();
            #endregion

            #region Count Property
            Console.WriteLine("Total Products: " + products.Count);
            #endregion

            #region Get All Keys and Values
            // Keys property
            foreach (var key in products.Keys)
            {
                Console.WriteLine("Key: " + key);
            }

            // Values property
            foreach (var val in products.Values)
            {
                Console.WriteLine("Value: " + val);
            }
            #endregion

            #region Safe Access using TryGetValue
            // TryGetValue avoids exception if key does not exist
            if (products.TryGetValue(1, out string value))
            {
                Console.WriteLine("Key 1 Value: " + value);
            }
            else
            {
                Console.WriteLine("Key 1 not found.");
            }
            #endregion
        }
    }
}
```

```
using System;
using System.Collections.Generic;

    public class Player
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Country { get; set; }
        public char Grade { get; set; }
        public int IccRank { get; set; }

        public override string ToString()
        {
            return $"{Id}: {Name} ({Country}) - Grade {Grade}, Rank {IccRank}";
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
```

```
            #region Create Dictionary with Complex Type
            // Key = PlayerId, Value = Player object
            Dictionary<int, Player> playersDict = new Dictionary<int, Player>();
            #endregion

            #region Add Elements
            playersDict.Add(1, new Player { Id = 1, Name = "Rohit Sharma", Country = "India",
Grade = 'A', IccRank = 1 });
            playersDict.Add(2, new Player { Id = 2, Name = "Virat Kohli", Country = "India", Grade
= 'A', IccRank = 2 });
            playersDict.Add(3, new Player { Id = 3, Name = "Bumrah", Country = "India", Grade =
'A', IccRank = 3 });
            playersDict.Add(4, new Player { Id = 4, Name = "Maxwell", Country = "Australia", Grade
= 'C', IccRank = 10 });
            #endregion

            #region Access Elements
            foreach (var kvp in playersDict)
            {
                Console.WriteLine($"Key={kvp.Key}, Value={kvp.Value}");
            }
            #endregion

            #region Update Element
            playersDict[2] = new Player { Id = 2, Name = "Virat Kohli", Country = "India", Grade =
'A', IccRank = 1 };
            Console.WriteLine("\nAfter Update: " + playersDict[2]);
            #endregion

            #region Safe Access using TryGetValue
            if (playersDict.TryGetValue(3, out Player foundPlayer))
            {
                Console.WriteLine("\nFound Player: " + foundPlayer);
            }
            else
            {
                Console.WriteLine("Player not found.");
            }
            #endregion

            #region Remove Element
            playersDict.Remove(4);
            Console.WriteLine("\nAfter Removing Player with Key=4:");
            foreach (var player in playersDict.Values)
            {
                Console.WriteLine(player);
            }
            #endregion
        }
    }
```

```
using System;
using System.Collections.Generic;

namespace _3.Stack
{
  class Program
  {
    static void Main(string[] args)
    {
```

```
#region What is Stack<T>?
/*
 * Stack<T> is a generic collection that works on the principle:
 * LIFO = Last In, First Out.
 *
 * Example in real life:
 * - Think of a stack of plates.
 *   The last plate kept is the first one you take out.
 *
 * Key Methods:
 * - Push() → Add element to top of stack
 * - Pop()  → Remove and return top element
 * - Peek() → Return top element (without removing)
 * - Contains() → Check if an item exists
 * - Count → Number of items
 * - Clear() → Remove all items
 */
#endregion

#region Create a Generic Stack
Stack<string> webHistory = new Stack<string>();
#endregion

#region Add Elements (Push)
webHistory.Push("www.google.com");
webHistory.Push("www.amazon.in");
webHistory.Push("www.youtube.com");
Console.WriteLine("Pages pushed into history.");
#endregion

#region Retrieve Elements
// Pop → removes and returns the top element
string lastVisited = webHistory.Pop();
Console.WriteLine("Last Visited (Pop): " + lastVisited);

// Peek → returns top element but does not remove
string currentPage = webHistory.Peek();
Console.WriteLine("Current Page (Peek): " + currentPage);
#endregion

#region Check Existence
Console.WriteLine("Contains www.youtube.com? " + webHistory.Contains("www.youtube.com"));
#endregion

#region Count
Console.WriteLine("Total Pages in History: " + webHistory.Count);
#endregion

#region Iterate Stack
Console.WriteLine("Browsing History (Top to Bottom):");
foreach (string page in webHistory)
{
   Console.WriteLine(page);
}
```

```
        #endregion

        #region Clear Stack
        webHistory.Clear();
        Console.WriteLine("History cleared. Count = " + webHistory.Count);
        #endregion
    }
  }
}
```

```
using System;
using System.Collections.Generic;

namespace _4.Queue
{
    class Program
    {
        static void Main(string[] args)
        {
            #region What is Queue<T>?
            /*
             * Queue<T> is a generic collection that works on the principle:
             * FIFO = First In, First Out.
             *
             * Example in real life:
             * - Think of a line at a ticket counter.
             *   The first person in line gets served first.
             *
             * Key Methods:
             * - Enqueue() → Add element to end of queue
             * - Dequeue() → Remove and return element from front
             * - Peek()    → Return front element (without removing)
             * - Contains() → Check if an item exists
             * - Clear() → Remove all items
             * - Count → Number of items
             */
            #endregion

            #region Create a Queue
            Queue<int> numbers = new Queue<int>();
            #endregion

            #region Add Elements (Enqueue)
            numbers.Enqueue(10);
            numbers.Enqueue(5);
            numbers.Enqueue(2);
            numbers.Enqueue(1);
            numbers.Enqueue(3);
            numbers.Enqueue(4);
            Console.WriteLine("Numbers enqueued into the queue.");
            #endregion

            #region Retrieve Elements
            // Dequeue → removes and returns the element from the front
            int first = numbers.Dequeue();
            Console.WriteLine("First Removed (Dequeue): " + first);

            // Peek → returns the element at the front without removing
            int front = numbers.Peek();
            Console.WriteLine("Current Front (Peek): " + front);
            #endregion
```

```
            #region Check Existence
            Console.WriteLine("Contains 4? " + numbers.Contains(4));
            #endregion

            #region Iterate Queue
            Console.WriteLine("Remaining Queue (Front to End):");
            foreach (int num in numbers)
            {
                Console.WriteLine(num);
            }
            #endregion

            #region Count
            Console.WriteLine("Total Elements in Queue: " + numbers.Count);
            #endregion

            #region Clear Queue
            numbers.Clear();
            Console.WriteLine("Queue cleared. Count = " + numbers.Count);
            #endregion
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace _5.HashSetExample
{
    class Program
    {
        static void Main(string[] args)
        {
            #region What is HashSet<T>?
            /*
             * HashSet<T>:
             * - A collection that stores UNIQUE values only (no duplicates).
             * - Fast lookups using hashing.
             * - Order of elements is NOT guaranteed.
             *
             * Example:
             * - Storing unique roll numbers of students.
             */
            #endregion

            #region Create a HashSet
            HashSet<int> numbers = new HashSet<int>();
            #endregion

            #region Add Elements
            bool added1 = numbers.Add(10);  // true
            bool added2 = numbers.Add(20);  // true
            bool added3 = numbers.Add(10);  // false (duplicate ignored)

            Console.WriteLine("10 added first time? " + added1);
            Console.WriteLine("10 added second time? " + added3);
            #endregion

            #region Access & Iterate
            Console.WriteLine("Elements in HashSet:");
            foreach (int num in numbers)
```

```
            {
                Console.WriteLine(num);
            }
            #endregion

            #region Check Existence
            Console.WriteLine("Contains 20? " + numbers.Contains(20));
            #endregion

            #region Remove Elements
            numbers.Remove(20);
            Console.WriteLine("After removing 20, count = " + numbers.Count);
            #endregion

            #region Set Operations
            HashSet<int> setA = new HashSet<int>() { 1, 2, 3, 4, 5 };
            HashSet<int> setB = new HashSet<int>() { 4, 5, 6, 7 };

            // Union
            setA.UnionWith(setB); // {1,2,3,4,5,6,7}
            Console.WriteLine("Union: " + string.Join(",", setA));

            // Reset setA
            setA = new HashSet<int>() { 1, 2, 3, 4, 5 };

            // Intersection
            setA.IntersectWith(setB); // {4,5}
            Console.WriteLine("Intersection: " + string.Join(",", setA));

            // Reset setA
            setA = new HashSet<int>() { 1, 2, 3, 4, 5 };

            // Difference (ExceptWith)
            setA.ExceptWith(setB); // {1,2,3}
            Console.WriteLine("ExceptWith: " + string.Join(",", setA));

            // Reset setA
            setA = new HashSet<int>() { 1, 2, 3, 4, 5 };

            // Symmetric Difference
            setA.SymmetricExceptWith(setB); // {1,2,3,6,7}
            Console.WriteLine("SymmetricExceptWith: " + string.Join(",", setA));
            #endregion

            #region Clear HashSet
            numbers.Clear();
            Console.WriteLine("HashSet cleared. Count = " + numbers.Count);
            #endregion
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace _6.SortedSetExample
{
    class Program
    {
        static void Main(string[] args)
        {
            #region What is SortedSet<T>?
```

```
/*
 * SortedSet<T>:
 * - Stores UNIQUE elements (like HashSet).
 * - Automatically keeps items SORTED (ascending by default).
 * - Can use a custom comparer for custom sorting.
 *
 * Real-world Example:
 * - A list of unique student roll numbers in ascending order.
 */
#endregion

#region Create a SortedSet
SortedSet<int> numbers = new SortedSet<int>();
#endregion

#region Add Elements
numbers.Add(50);
numbers.Add(10);
numbers.Add(20);
numbers.Add(40);
numbers.Add(30);
numbers.Add(20); // Duplicate → ignored

Console.WriteLine("✅ SortedSet Elements (Auto Sorted):");
foreach (var num in numbers)
{
    Console.WriteLine(num);
}
// Output:
// 10
// 20
// 30
// 40
// 50
#endregion

#region Properties (Min & Max)
Console.WriteLine("\nMin: " + numbers.Min);  // 10
Console.WriteLine("Max: " + numbers.Max);    // 50
#endregion

#region Check Existence
Console.WriteLine("\nContains 30? " + numbers.Contains(30)); // True
Console.WriteLine("Contains 99? " + numbers.Contains(99));   // False
#endregion

#region Remove Elements
numbers.Remove(20);
Console.WriteLine("\nAfter removing 20: " + string.Join(", ", numbers));
// Output: 10, 30, 40, 50
#endregion

#region Get Range (View Between)
var range = numbers.GetViewBetween(15, 40);
Console.WriteLine("\nRange (15-40): " + string.Join(", ", range));
// Output: 30, 40
#endregion

#region Set Operations
SortedSet<int> setA = new SortedSet<int>() { 1, 2, 3, 4, 5 };
SortedSet<int> setB = new SortedSet<int>() { 4, 5, 6, 7 };
```

```
            // Union
            setA.UnionWith(setB);
            Console.WriteLine("\nUnion: " + string.Join(",", setA));
            // Output: 1,2,3,4,5,6,7

            // Reset setA
            setA = new SortedSet<int>() { 1, 2, 3, 4, 5 };

            // Intersection
            setA.IntersectWith(setB);
            Console.WriteLine("Intersection: " + string.Join(",", setA));
            // Output: 4,5

            // Reset setA
            setA = new SortedSet<int>() { 1, 2, 3, 4, 5 };

            // Difference
            setA.ExceptWith(setB);
            Console.WriteLine("ExceptWith: " + string.Join(",", setA));
            // Output: 1,2,3

            // Reset setA
            setA = new SortedSet<int>() { 1, 2, 3, 4, 5 };

            // Symmetric Difference
            setA.SymmetricExceptWith(setB);
            Console.WriteLine("SymmetricExceptWith: " + string.Join(",", setA));
            // Output: 1,2,3,6,7
            #endregion

            #region Clear
            numbers.Clear();
            Console.WriteLine("\nSortedSet cleared. Count = " + numbers.Count);
            // Output: 0
            #endregion
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace _7.LinkedListExample
{
    class Program
    {
        static void Main(string[] args)
        {
            #region What is LinkedList<T>?
            /*
             * LinkedList<T>:
             * - Doubly-linked list storing nodes with pointers to previous & next nodes.
             * - Fast insertion/removal anywhere in the list.
             * - Maintains order of elements.
             */
            #endregion

            #region Create LinkedList
            LinkedList<string> fruits = new LinkedList<string>();
            #endregion

            #region Add Elements
```

```
            fruits.AddLast("Apple");    // Add at end
            fruits.AddLast("Banana");
            fruits.AddFirst("Mango");   // Add at start
            fruits.AddLast("Orange");

            Console.WriteLine("LinkedList after AddFirst & AddLast:");
            foreach (var fruit in fruits)
                Console.WriteLine(fruit);
            // Output: Mango, Apple, Banana, Orange
            #endregion

            #region Add Before / After
            var node = fruits.Find("Banana");
            fruits.AddBefore(node, "Pineapple");
            fruits.AddAfter(node, "Grapes");

            Console.WriteLine("\nLinkedList after AddBefore & AddAfter:");
            foreach (var fruit in fruits)
                Console.WriteLine(fruit);
            // Output: Mango, Apple, Pineapple, Banana, Grapes, Orange
            #endregion

            #region Remove Elements
            fruits.Remove("Apple");        // Remove by value
            fruits.RemoveFirst();          // Remove first node
            fruits.RemoveLast();           // Remove last node

            Console.WriteLine("\nLinkedList after Remove operations:");
            foreach (var fruit in fruits)
                Console.WriteLine(fruit);
            // Output: Pineapple, Banana, Grapes
            #endregion

            #region Check Existence
            Console.WriteLine("\nContains 'Banana'? " + fruits.Contains("Banana")); // True
            Console.WriteLine("Contains 'Mango'? " + fruits.Contains("Mango"));     // False
            #endregion

            #region Properties
            Console.WriteLine("\nFirst Node: " + fruits.First.Value); // Pineapple
            Console.WriteLine("Last Node: " + fruits.Last.Value);     // Grapes
            Console.WriteLine("Count: " + fruits.Count);             // 3
            #endregion

            #region Clear LinkedList
            fruits.Clear();
            Console.WriteLine("\nLinkedList cleared. Count = " + fruits.Count);
            // Output: 0
            #endregion
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace _8.SortedListExample
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
            #region What is SortedList<TKey,TValue>?
            /*
             * SortedList<TKey,TValue>:
             * - Stores key-value pairs like Dictionary.
             * - Keys are UNIQUE, values can be duplicates.
             * - Keys are automatically SORTED (ascending by default).
             * - Uses an internal array → index-based access is fast.
             * - Insertion/Deletion in large lists is slower compared to SortedDictionary.
             */
            #endregion

            #region Create SortedList
            SortedList<int, string> products = new SortedList<int, string>();
            #endregion

            #region Add Elements
            // Add key-value pairs
            products.Add(5, "Samsung");
            products.Add(2, "Redmi");
            products.Add(1, "OnePlus");
            products.Add(4, "Nothing");

            Console.WriteLine("✅ SortedList Elements (Auto Sorted by Key):");
            foreach (var kvp in products)
            {
                Console.WriteLine($"Key={kvp.Key}, Value={kvp.Value}");
            }
            // Output:
            // Key=1, Value=OnePlus
            // Key=2, Value=Redmi
            // Key=4, Value=Nothing
            // Key=5, Value=Samsung
            #endregion

            #region Access by Index
            // You can access keys/values by index because it's internally array-based
            Console.WriteLine($"\nElement at index 2: Key={products.Keys[2]},
Value={products.Values[2]}");
            // Output: Key=4, Value=Nothing
            #endregion

            #region Check Existence
            Console.WriteLine("Contains Key 2? " + products.ContainsKey(2));     // True
            Console.WriteLine("Contains Value 'Apple'? " + products.ContainsValue("Apple")); //
False
            #endregion

            #region Remove Elements
            products.Remove(4); // Remove element by key
            Console.WriteLine("\nAfter removing key=4:");
            foreach (var kvp in products)
                Console.WriteLine($"Key={kvp.Key}, Value={kvp.Value}");
            #endregion

            #region Count & Clear
            Console.WriteLine($"\nTotal Elements: {products.Count}"); // 3
            products.Clear(); // Remove all elements
            Console.WriteLine("After Clear, Count = " + products.Count); // 0
            #endregion
        }
    }
}
```

```
using System;
using System.Collections.Generic;

namespace _8.SortedDictionaryExample
{
    class Program
    {
        static void Main(string[] args)
        {
            #region What is SortedDictionary<TKey,TValue>?
            /*
             * SortedDictionary<TKey,TValue>:
             * - Stores key-value pairs like Dictionary.
             * - Keys are UNIQUE, values can be duplicates.
             * - Keys are automatically SORTED (ascending by default).
             * - Uses a Binary Search Tree internally → better for frequent insertions/deletions.
             * - Does NOT support access by index.
             */
            #endregion

            #region Create SortedDictionary
            SortedDictionary<int, string> products = new SortedDictionary<int, string>();
            #endregion

            #region Add Elements
            products.Add(5, "Samsung");
            products.Add(2, "Redmi");
            products.Add(1, "OnePlus");
            products.Add(4, "Nothing");

            Console.WriteLine("✅ SortedDictionary Elements (Auto Sorted by Key):");
            foreach (var kvp in products)
            {
                Console.WriteLine($"Key={kvp.Key}, Value={kvp.Value}");
            }
            // Output:
            // Key=1, Value=OnePlus
            // Key=2, Value=Redmi
            // Key=4, Value=Nothing
            // Key=5, Value=Samsung
            #endregion

            #region Check Existence
            Console.WriteLine("\nContains Key 2? " + products.ContainsKey(2));   // True
            Console.WriteLine("Contains Value 'Apple'? " + products.ContainsValue("Apple")); //
False
            #endregion

            #region Remove & Count
            products.Remove(4); // Remove element by key
            Console.WriteLine("\nAfter removing key=4:");
            foreach (var kvp in products)
                Console.WriteLine($"Key={kvp.Key}, Value={kvp.Value}");

            Console.WriteLine($"\nTotal Elements: {products.Count}"); // 3
            #endregion

            #region Clear
            products.Clear(); // Remove all elements
            Console.WriteLine("After Clear, Count = " + products.Count); // 0
            #endregion
        }}}
```

```
   using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace _9.CollectionConversions
{
    class Program
    {
        static void Main(string[] args)
        {
            #region 1  Array to List
            int[] numbersArray = { 1, 2, 3, 4, 5 };
            List<int> numbersList = numbersArray.ToList(); // Using LINQ
            Console.WriteLine("Array to List:");
            numbersList.ForEach(n => Console.Write(n + " "));
            Console.WriteLine(); // Output: 1 2 3 4 5
            #endregion

            #region 2  List to Array
            int[] arrayFromList = numbersList.ToArray();
            Console.WriteLine("\nList to Array:");
            foreach (var n in arrayFromList)
                Console.Write(n + " ");
            Console.WriteLine(); // Output: 1 2 3 4 5
            #endregion

            #region 3  Array to ArrayList
            ArrayList arrayList = new ArrayList(numbersArray); // Non-generic collection
            Console.WriteLine("\nArray to ArrayList:");
            foreach (var n in arrayList)
                Console.Write(n + " ");
            Console.WriteLine(); // Output: 1 2 3 4 5
            #endregion

            #region 4  List to Dictionary
            // Suppose list has tuples or key-value data
            List<(int Id, string Name)> listData = new List<(int, string)>
            {
                (1, "Alice"),
                (2, "Bob"),
                (3, "Charlie")
            };

            Dictionary<int, string> dictFromList = listData.ToDictionary(x => x.Id, x => x.Name);
            Console.WriteLine("\nList to Dictionary:");
            foreach (var kvp in dictFromList)
                Console.WriteLine($"Key={kvp.Key}, Value={kvp.Value}");
            /*
            Output:
            Key=1, Value=Alice
            Key=2, Value=Bob
            Key=3, Value=Charlie
            */
            #endregion

            #region 5  Array to Dictionary
            string[] names = { "Alice", "Bob", "Charlie" };
            // Using index as key
            Dictionary<int, string> dictFromArray = names.Select((value, index) => new { index,
value })
                                                         .ToDictionary(x => x.index, x => x.value);
```

```
            Console.WriteLine("\nArray to Dictionary:");
            foreach (var kvp in dictFromArray)
                Console.WriteLine($"Key={kvp.Key}, Value={kvp.Value}");
            /*
            Output:
            Key=0, Value=Alice
            Key=1, Value=Bob
            Key=2, Value=Charlie
            */
            #endregion

            #region 6  Dictionary to List
            Dictionary<int, string> dict = new Dictionary<int, string>
            {
                {1,"One"},{2,"Two"},{3,"Three"}
            };

            // Convert keys to List
            List<int> keysList = dict.Keys.ToList();
            // Convert values to List
            List<string> valuesList = dict.Values.ToList();

            Console.WriteLine("\nDictionary Keys to List:");
            keysList.ForEach(k => Console.Write(k + " ")); // Output: 1 2 3
            Console.WriteLine("\nDictionary Values to List:");
            valuesList.ForEach(v => Console.Write(v + " ")); // Output: One Two Three
            #endregion

            #region 7  Dictionary to Array
            int[] keysArray = dict.Keys.ToArray();
            string[] valuesArray = dict.Values.ToArray();

            Console.WriteLine("\n\nDictionary Keys to Array:");
            foreach (var k in keysArray) Console.Write(k + " "); // Output: 1 2 3
            Console.WriteLine("\nDictionary Values to Array:");
            foreach (var v in valuesArray) Console.Write(v + " "); // Output: One Two Three
            #endregion
        }
    }
}
```