HashSet

A set is a data structure that stores unique elements of the same type in a sorted order. Each value is a key, which means that we access each value using the value itself. With arrays, on the other hand, we access each value by its position in the container (the index).

Set is the general interface to a set-like collection, while HashSet is a specific implementation of the Set interface (which uses hash codes, hence the name). Set is a parent interface of all set classes like TreeSet. **HashSet** is implementation of Set Interface which does not allow duplicate value.

Examples:

- Spell Checker that checks words as a user type them needs to extremely fast, so it uses a hashset of real words to check if a given word is real or not

Operations In HashSet

Insert

- adds item to the indicated place/cell
- the process is very fast because it only includes one step
- every data inserted is placed in the first vacant cell

Search

- Another process that the algorithm carries out
- Moves methodically downwards to search for the item

Delete

- The algorithm must first locate the item to delete it
- The deleted item causes hole in the array
- Hole – one or more cells that have filled cells above them

HashSet in C#

In C#, HashSet is an unordered collection of unique elements. This collection is introduced in *.NET 3.5*. It supports the implementation of sets and uses the hash table for storage. This collection is of the generic type collection and it is defined under *System.Collections.Generic* namespace. It is generally used when we want to prevent duplicate elements from being placed in the collection. The performance of the HashSet is much better in comparison to the list.

Important Points:

- The HashSet class implements the *ICollection*, *IEnumerable*, *IReadOnlyCollection*, *ISet*, *IEnumerable*, *IDes erializationCallback*, and *ISerializable* interfaces.

- In HashSet, the order of the element is not defined. You cannot sort the elements of HashSet.
- In HashSet, the elements must be unique.

- In HashSet, duplicate elements are not allowed.
- Is provides many mathematical set operations, such as intersection, union, and difference.

- The capacity of a HashSet is the number of elements it can hold.
- A HashSet is a dynamic collection means the size of the HashSet is automatically increased when the new elements are added.
- In HashSet, you can only store the same type of elements.

The HashSet class provides *7 different types of constructors* which are used to create a HashSet, here we only use *HashSet()*, constructor.

HashSet(): It is used to create an instance of the HashSet class that is empty and uses the default equality comparer for the set type.

Step 1: Include *System.Collections.Generic* namespace in your program with the help of *using* keyword:

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

Step 2: Create a HashSet using the HashSet class as shown below:

```
HashSet<Type_of_hashset> Hashset_name = new HashSet<Type_of_hashset>();
```

Step 3: If you want to add elements in your HashSet, then use *Add()* method to add elements in your HashSet. And you can also store elements in your HashSet using collection initializer.

Step 4: The elements of HashSet is accessed by using a *foreach* loop. As shown in the below example.

Example program 1

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

class HashSet {
  // Main Method
  static public void Main()
  {
    // Creating HashSet
    // Using HashSet class
    HashSet<string> myhash1 = new HashSet<string>();

    // Add the elements in HashSet
    // Using Add method
    myhash1.Add("Math");
    myhash1.Add("English");
    myhash1.Add("Science");
    myhash1.Add("Filipino");
    myhash1.Add("History");
    Console.WriteLine("Elements of myhash1:");

    // Accessing elements of HashSet
    // Using foreach loop
    foreach(var val in myhash1)
    {
      Console.WriteLine(val);
    }

    // Creating another HashSet
    // using collection initializer
    // to initialize HashSet
    HashSet<int> myhash2 = new HashSet<int>() {90,95,99,83,86};

    // Display elements of myhash2
    Console.WriteLine("Elements of myhash2:");
    foreach(var value in myhash2)
    {
      Console.WriteLine(value);
```

```
Elements of myhash1:
Math
English
Science
Filipino
History
Elements of myhash2:
90
95
99
83
86
```

How to Remove HashSet in C#?

In HashSet, you are allowed to remove elements from the HashSet. HashSet<T> class provides three different methods to remove elements and the **methods are:**

- **Remove(T):** This method is used to remove the specified element from a HashSet object.
- **RemoveWhere(Predicate):** This method is used to remove all elements that match the conditions defined by the specified predicate from a HashSet collection.
- **Clear:** This method is used to remove all elements from a HashSet object.

Hashset operations

HashSet class also provides some methods that are used to perform different operations on sets . The method is used in a situation where we want to prevent duplicates from being inserted in the collection.

- **UnionWith(IEnumerable):** This method is used to modify the current HashSet object to contain all elements that are present in itself, the specified collection, or both.
- **IntersectWith(IEnumerable):** This method is used to modify the current HashSet object to contain only elements that are present in that object and in the specified collection.
- **ExceptWith(IEnumerable): This method is used to remove all elements in the specified collection from the current HashSet object.**

Example Program Using UnionWith(IEnumerable)

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

class RemoveElementsFromHash {

    // Main Method
    static public void Main()
    {
        // Creating HashSet
        // Using HashSet class
        HashSet<string> myhash = new HashSet<string>();

        // Add the elements in HashSet
        // Using Add method
        myhash.Add("Math");
        myhash.Add("English");
        myhash.Add("Filipino");
        myhash.Add("History");
        myhash.Add("Science");

        // Before using Remove method
        Console.WriteLine("Total number of elements present (Before Removal)"+
                          " in myhash: {0}", myhash.Count);

        // Remove element from HashSet
        // Using Remove method
        myhash.Remove("Science");

        // After using Remove method
        Console.WriteLine("Total number of elements present (After Removal)"+
                          " in myhash: {0}", myhash.Count);

        // Remove all elements from HashSet
        // Using Clear method
        myhash.Clear();
        Console.WriteLine("Total number of elements present"+
                           " in myhash:{0}", myhash.Count);
        Console.Read();
    }
}
```

```
Total number of elements present (Before Removal) in myhash: 5
Total number of elements present (After Removal) in myhash: 4
Total number of elements present in myhash:0
```

Example Program using **IntersectWith(IEnumerable):**

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

class Intersection {
    // Main Method
    static public void Main()
    {
        // Creating HashSet
        // Using HashSet class
        HashSet<string> myhash1 = new HashSet<string>();

        // Add the elements in HashSet
        // Using Add method
        myhash1.Add("Discrete Math");
        myhash1.Add("Networking");
        myhash1.Add("Data Structure");
        myhash1.Add("OOP");
        myhash1.Add("DBMS");

        // Creating another HashSet
        // Using HashSet class
        HashSet<string> myhash2 = new HashSet<string>();

        // Add the elements in HashSet
        // Using Add method
        myhash2.Add("OOP");
        myhash2.Add("DBMS");
        myhash2.Add("Data Structure");
        myhash2.Add("Thesis Writing");

        // Using IntersectWith method
        myhash1.IntersectWith(myhash2);
        foreach(var ele in myhash1)
        {
            Console.WriteLine(ele);
            Console.ReadKey();
        }
    }
}
```

```
Data Structure
OOP
DBMS
```

Example Program Using **ExceptWith(IEnumerable)**:

```
using System;
using System.Collections.Generic;
class Exception {
    // Main Method
    static public void Main()
    {
        // Creating HashSet
        // Using HashSet class
        HashSet<string> myhash1 = new HashSet<string>();

        // Add the elements in HashSet
        // Using Add method
        myhash1.Add("Discrete Math");
        myhash1.Add("Data Structure");
        myhash1.Add("OOP");
        myhash1.Add("Networking");
        myhash1.Add("Platform Technologies");

        // Creating another HashSet
        // Using HashSet class
        HashSet<string> myhash2 = new HashSet<string>();

        // Add the elements in HashSet
        // Using Add method
        myhash2.Add("Intermediate Programming");
        myhash2.Add("Multimedia Systems");
        myhash2.Add("Discrete Math");
        myhash2.Add("Data Structure");

        // Using ExceptWith method
        myhash1.ExceptWith(myhash2);
        foreach(var ele in myhash1)
        {
            Console.WriteLine(ele);
            Console.ReadKey();
        }
    }
}
```

```
OOP
Networking
Platform Technologies
```