

Disadvantages/Limitations of Arrays

1. **We can not enter elements of different type : Advantage** , Type safe
2. Fixed Memory

Num[10]

Memory is allocated at compile time, memory can be wasted

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 5 | 6 | 8 | 90 | 7 |  |  |  |  |

1. we cannot change array size dynamically
2. insertion / deletion is time consuming because it requires lots of element reshuffling

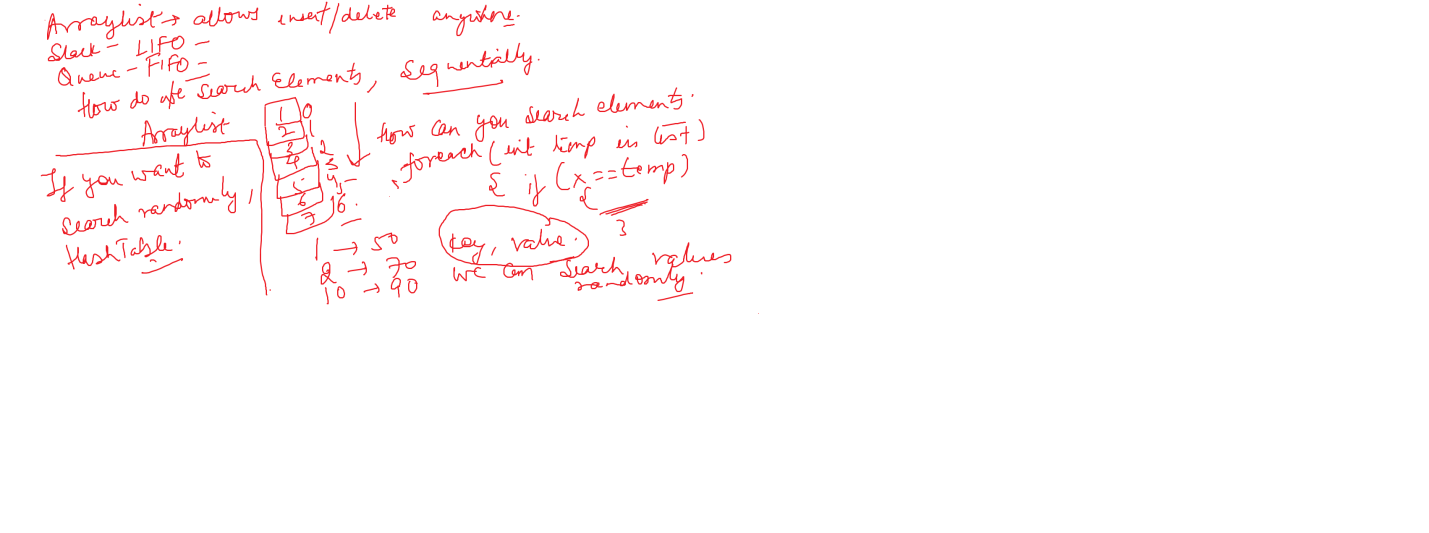
Q. WAP to insert some elements in Array

Q. WAP to delete some elements in Array

TO solve problems posed by Arrays , we use collection

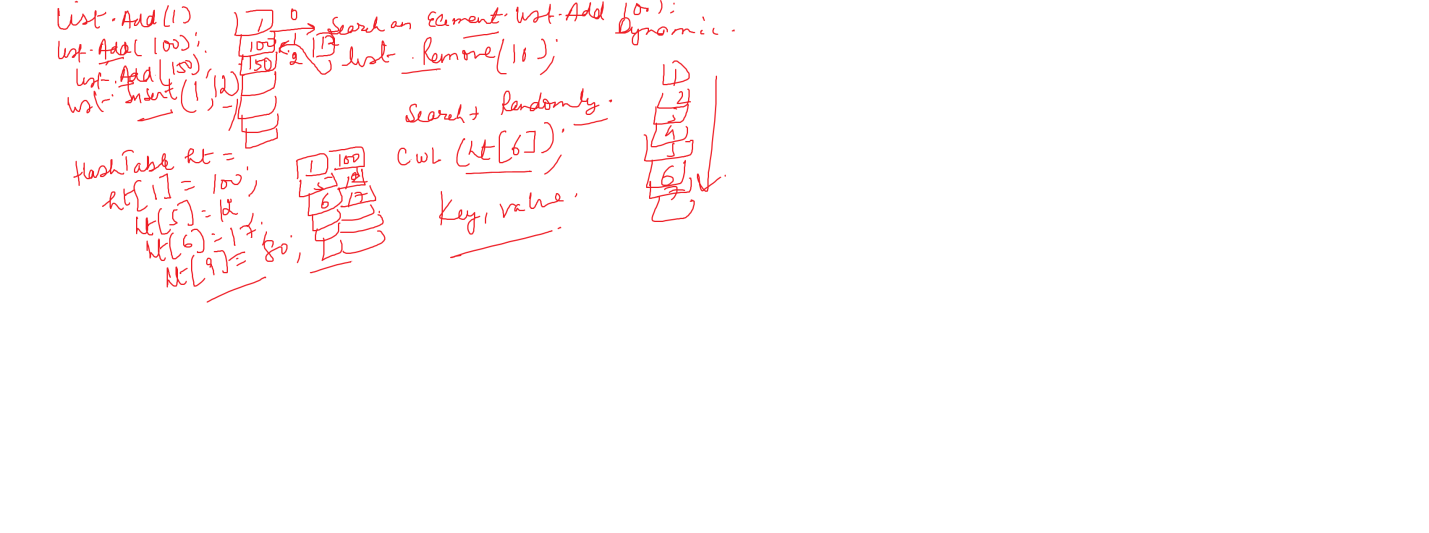
Collection is a structure in which we can store elements of different types

To use collections , we have to use System.Collections namespace



Collections could be

* ArrayList > insertion / deletion can be done anywhere
* Stack > Follows LIFO , insertion / deletion both are done at end
* Queue > Follows FIFO , insertion at end , deletion from beginning
* HashSet > Here we use Key, value
* SortedList



// ArrayList

using System;

using System.Collections;

class CollectionsDemo

{

static void Main()

{

// Array

//int[] num = new int[10];

// Fixed Memory is solved

ArrayList list = new ArrayList();

list.Add(10);

list.Add(20);

list.Add(30);

for(int i=0;i<list.Count;i++)

Console.WriteLine(list[i]);

list.Insert(0, 100);

Console.WriteLine("After inserting at position 0");

for (int i = 0; i < list.Count; i++)

Console.WriteLine(list[i]);

list.Remove(10);

Console.WriteLine("After Deleting 10");

for (int i = 0; i < list.Count; i++)

Console.WriteLine(list[i]);

}

}

To iterate thru collection / or an array, we also use foreach loop

foreach(int x in stack)

Console.WriteLine(x);

Foreach( type rangevaribale in collection/array name)

{

Console.Write(rangevaribale);

}

Stack

using System;

using System.Collections;

class CollectionsDemo

{

static void Main()

{

// Array

//int[] num = new int[10];

// Fixed Memory is solved

// ArrayList list = new ArrayList();

// list.Add(10);

// list.Add(20);

// list.Add(30);

//for(int i=0;i<list.Count;i++)

// Console.WriteLine(list[i]);

// list.Insert(0, 100);

// Console.WriteLine("After inserting at position 0");

// for (int i = 0; i < list.Count; i++)

// Console.WriteLine(list[i]);

// list.Remove(10);

// Console.WriteLine("After Deleting 10");

// for (int i = 0; i < list.Count; i++)

// Console.WriteLine(list[i]);

Stack stack = new Stack();

stack.Push(10);

stack.Push(20);

stack.Push(30);

stack.Push(40);

foreach (int x in stack)

Console.WriteLine(x);

stack.Pop();

Console.WriteLine("Elements after Deletion from stack");

foreach (int x in stack)

Console.WriteLine(x);

}

}

Queue

using System;

using System.Collections;

class CollectionsDemo

{

static void Main()

{

// Array

//int[] num = new int[10];

// Fixed Memory is solved

// ArrayList list = new ArrayList();

// list.Add(10);

// list.Add(20);

// list.Add(30);

//for(int i=0;i<list.Count;i++)

// Console.WriteLine(list[i]);

// list.Insert(0, 100);

// Console.WriteLine("After inserting at position 0");

// for (int i = 0; i < list.Count; i++)

// Console.WriteLine(list[i]);

// list.Remove(10);

// Console.WriteLine("After Deleting 10");

// for (int i = 0; i < list.Count; i++)

// Console.WriteLine(list[i]);

Queue queue = new Queue();

queue.Enqueue(10);

queue.Enqueue(20);

queue.Enqueue(30);

queue.Enqueue(40);

foreach (int x in queue)

Console.WriteLine(x);

queue.Dequeue();

Console.WriteLine("Elements after Deletion from queue");

foreach (int x in queue)

Console.WriteLine(x);

}

}

Boxing / Unboxing

Boxing > Convert from value type to reference type

UnBoxing > Convert from reference type to value type

using System;

using System.Collections;

class CollectionsDemo

{

static void Main()

{

// Array

//int[] num = new int[10];

// Fixed Memory is solved

ArrayList list = new ArrayList();

list.Add(10);

list.Add(20);

list.Add(30);

list.Add("Ajay");

list.Add(90.98);

//list.Add()

foreach(var x in list)

Console.WriteLine(x);

list.Insert(0, 100);

Console.WriteLine("After inserting at position 0");

foreach (var x in list)

Console.WriteLine(x);

list.Remove(10);

Console.WriteLine("After Deleting 10");

foreach (var x in list)

Console.WriteLine(x);

Queue queue = new Queue();

queue.Enqueue(10);

queue.Enqueue(20);

queue.Enqueue(30);

queue.Enqueue(40);

queue.Enqueue("Ajay");

foreach (var x in queue)

Console.WriteLine(x);

queue.Dequeue();

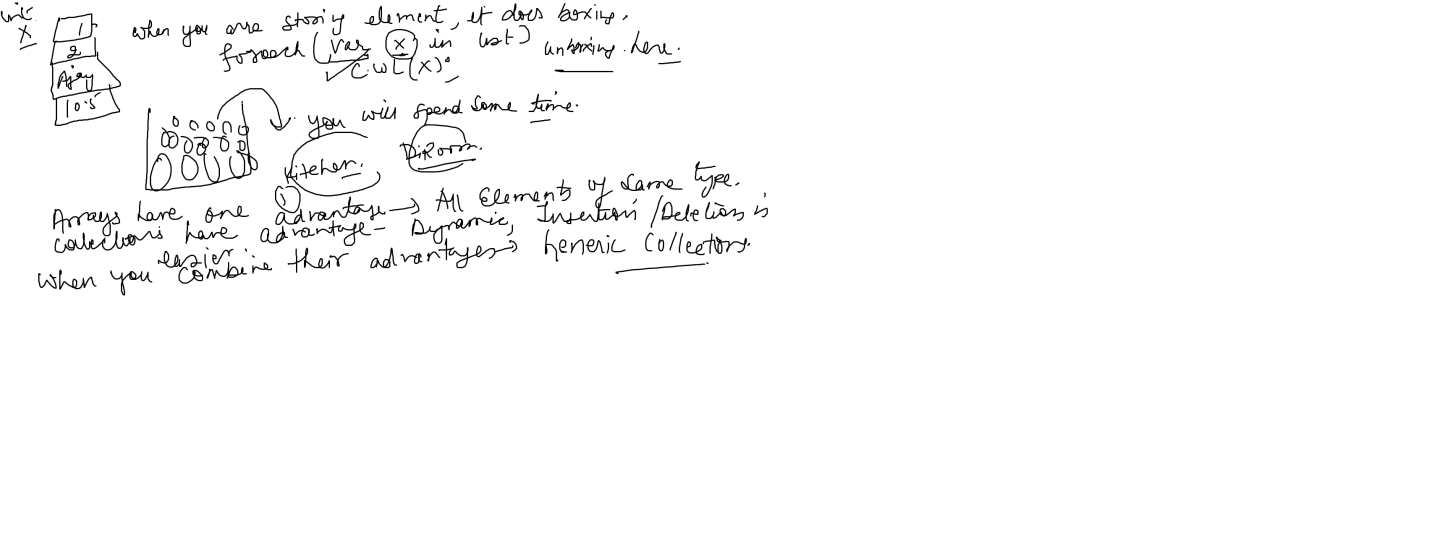
Console.WriteLine("Elements after Deletion from queue");

foreach (var x in queue)

Console.WriteLine(x);

}

}



Arrays : Advantage : Elements are of same type

Collections : Dynamic Size, memory is not wasted

Insertion / Deletion is not time consuming

When we combine advantages of both of them , we get Generic Collections

Generic means , type is known at compile time

ArrayList list = new ArrayList();

List<int> list = new List<int>();

using System;

using System.Collections.Generic;

class GenericCollectionsDemo

{

static void Main()

{

List<int> list = new List<int>();

list.Add(1);

list.Add(2);

list.Add(3);

list.Add(4);

foreach(int x in list)

Console.WriteLine(x);

list.Insert(0, 100);

Stack<string> stack = new Stack<string>();

stack.Push("Ajay");

stack.Push("Ajay");

stack.Push("Ajay");

stack.Push("Ajay");

foreach(string x in stack)

Console.WriteLine(x);

}

}

Hashtable hashtable = new Hashtable();

hashtable[0] = 90;

hashtable[2] = 89;

hashtable[3] = 78;

for (int i = 0; i <= hashtable.Keys.Count; i++)

Console.WriteLine(hashtable[i]);

//foreach (var x in hashtable)

// Console.WriteLine(x);

hashtable["Ajay"] = 90;

hashtable["Deepak"] = "Delhi";

Console.WriteLine(hashtable["Ajay"]);

Dictionary<int, int> keyValues = new Dictionary<int, int>();

keyValues[9] = 90;

