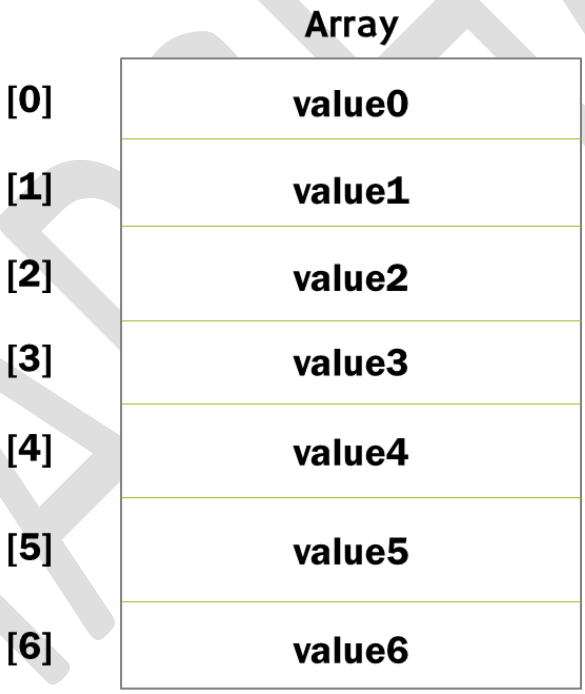
35

Arrays

Introducing Arrays

- Array is a group of multiple values of same type.
- Arrays are stored in continuous-memory-locations in 'heap'.



Array

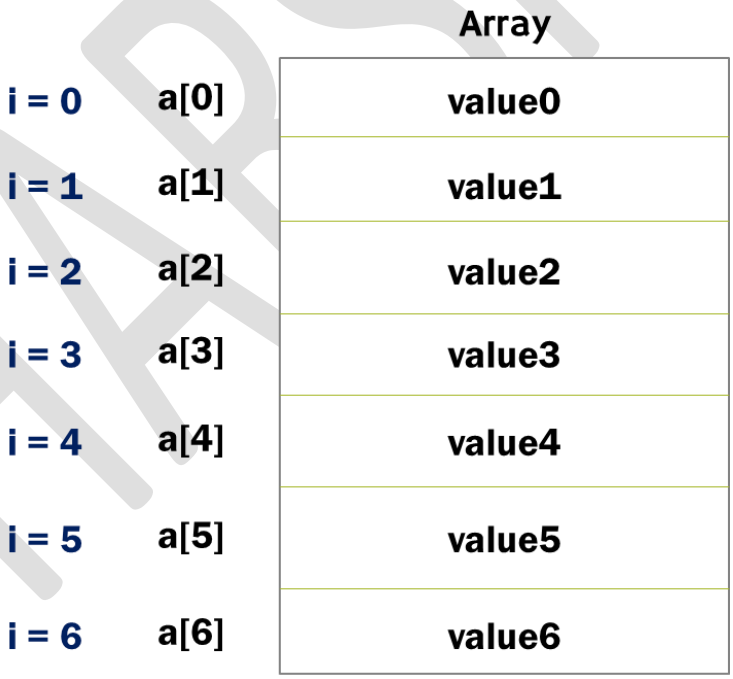
```
type[ ]  arrayReferenceVariableName = new  type[ size ];
```

- Each value of array is called "element".

- All the elements are stored in continuous memory locations.
- The address of first element of the array will be stored in the "array reference variable".
- The "Length" property stores count of elements of array. Index starts from 0 (zero).
- Arrays are treated as objects of "System.Array" class; so arrays are stored in heap; its address (first element's address) is stored in reference variable at stack.

Array 'for' loop

- For loop starts with an "initialization"; checks the condition; executes the loop body and then performs incrementation or decrementation;
- Use "for loop" to read elements from an array, based on index.



Array - For Loop

```
for (int i = 0; i < arrayRefVariable.Length; i++)
{
        arrayRefVariable[i]
}
```

Pros

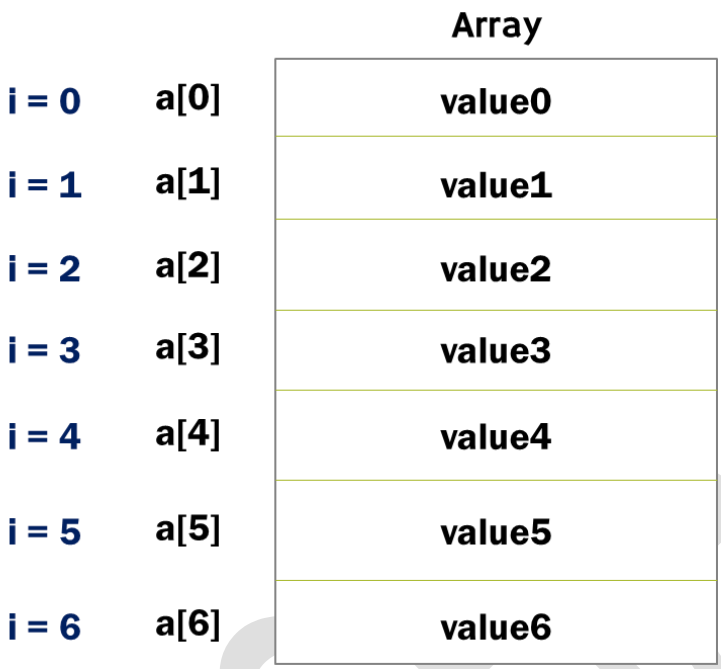
- You can read any part of the array (all elements, start from specific element, end with specific element).
- You can read array elements in reverse.

Cons

- A bit complex syntax.

Array 'foreach' loop

- Foreach loop starts contains a "iteration variable"; reads each value of an array or collection and assigns to the "iteration variable", till end of the array / collection.
- "Foreach loop" is not based on index; it is based on sequence.



Array - ForEach Loop

```
foreach (DataType iterationVariable in arrayVariable)
{
        iterationVariable
}
```

Pros

- Simplified Syntax
- Easy to use with arrays and collections.
- It internally uses "Iterators".

Cons

- Slower performance, due to it treats everything as a collection.

- It can't be used to execute repeatedly, without arrays or collections.
- It can't read part of array / collection, or read array / collection reverse.

System.Array class

- Every array is treated as an object for System.Array class.
- The System.Array class provides a set of properties and methods for every array.

Properties

- Length

Methods

- IndexOf
- BinarySearch
- Clear
- Resize
- Sort
- Reverse
- CopyTo
- Clone

Array – IndexOf( ) method

- This method searches the array for the given value.
    - If the value is found, it returns its index.

- If the value is not found, it returns -1.

Array - IndexOf() method

```
static int Array.IndexOf( System.Array array, object value)
```

Array - IndexOf() - Example

```
Array.IndexOf(value3) = 3
```

- The "IndexOf" method performs linear search. That means it searches all the elements of an array, until the search value is found. When the search value is found in the array, it stops searching and returns its index.
- The linear search has good performance, if the array is small. But if the array is larger, Binary search is recommended to improve the performance

Parameters

- **array:** This parameter represents the array, in which you want to search.
- **value:** This parameter represents the actual value that is to be searched.
- **startIndex:** This parameter represents the start index, from where the search should be started.

Array – BinarySearch( ) method

- This method searches the array for the given value.
  - If the value is found, it returns its index.

- If the value is not found, it returns -1.

Array - BinarySearch() method

```
static int Array.BinarySearch( System.Array array, object value)
```

Array - BinarySearch() - Example

```
Array.BinarySearch(value3) = 3
```

- The "Binary Search" requires an array, which is already sorted.
  - On unsorted arrays, binary search is not possible.
- It directly goes to the middle of the array (array size / 2), and checks that item is less than / greater than the search value.
- If that item is greater than the search value, it searches only in the first half of the array.
- If that item is less than the search value, it searches only in the second half of the array.
- Thus it searches only half of the array. So in this way, it saves performance

Parameters

- **array:** This parameter represents the array, in which you want to search.
- **value:** This parameter represents the actual value that is to be searched

Array – Clear( ) method

› This method starts with the given index and sets all the "length" no. of elements to zero (0).

Array - Clear( ) method
static void Array.Clear(System.Array array, int index, int length)

Clear( ) - Example
Array.Clear(a, 2, 3)

Array after Clear( ) method:

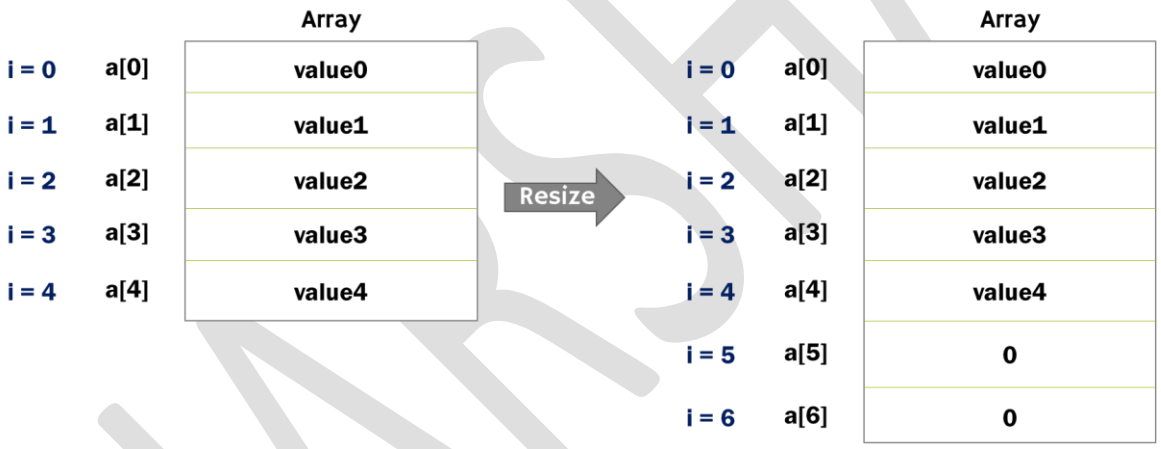
Array
value0
value1
0
0
0
value5
value6

Parameters

- **array:** This parameter represents the array, in which you want to clear the elements.
- **index:** This parameter represents the index, from which clearing process is to be started.
- **length:** This parameter represents the no. of elements that are to be cleared.

Array – Resize( ) method

- This method increases / decreases size of the array.



Array - Resize() method

```
static void Array.Resize(ref System.Array array, int newSize)
```

Array - Resize() - Example

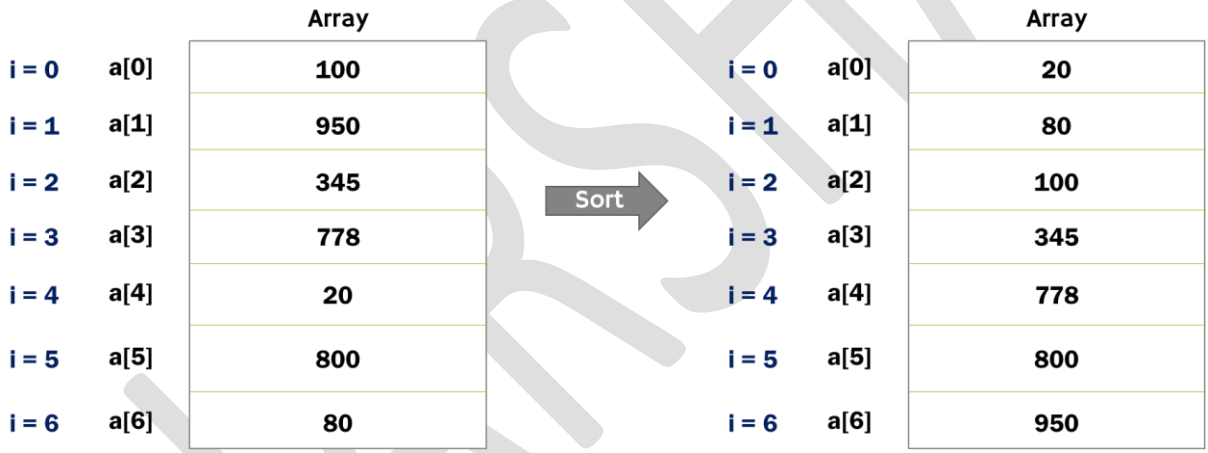
```
Array.Resize(a, 6)
```

Parameters

- **array:** This parameter represents the array, which you want to resize.
- **newSize:** This parameter represents the new size of the array, how many elements you want to store in the array. It can be less than or greater than the current size.

Array – Sort( ) method

- This method sorts the array in ascending order.



Array - Sort( ) method

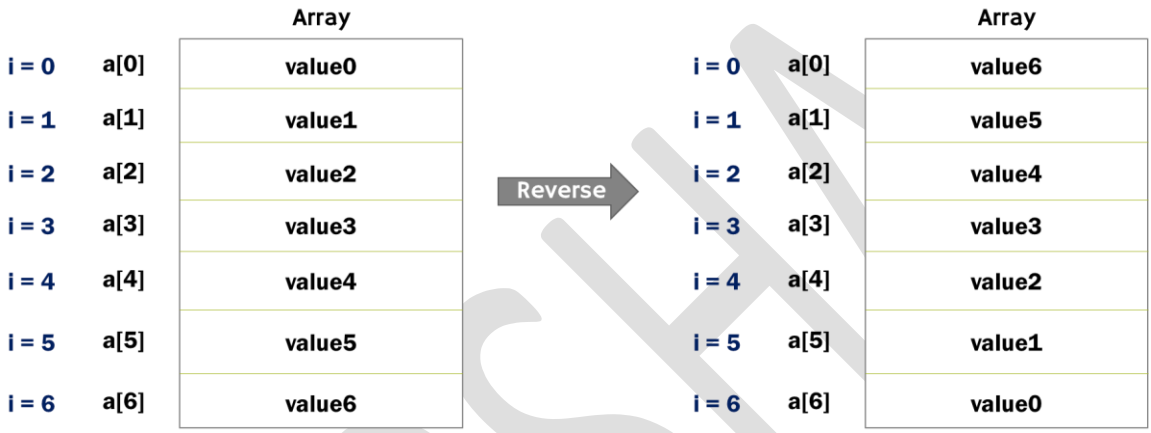
```
static void Array.Sort( System.Array array )
```

Array - Sort( ) - Example

```
Array.Sort(a)
```

Array – Reverse( ) method

› This method reverses the array.



Array - Reverse( ) method

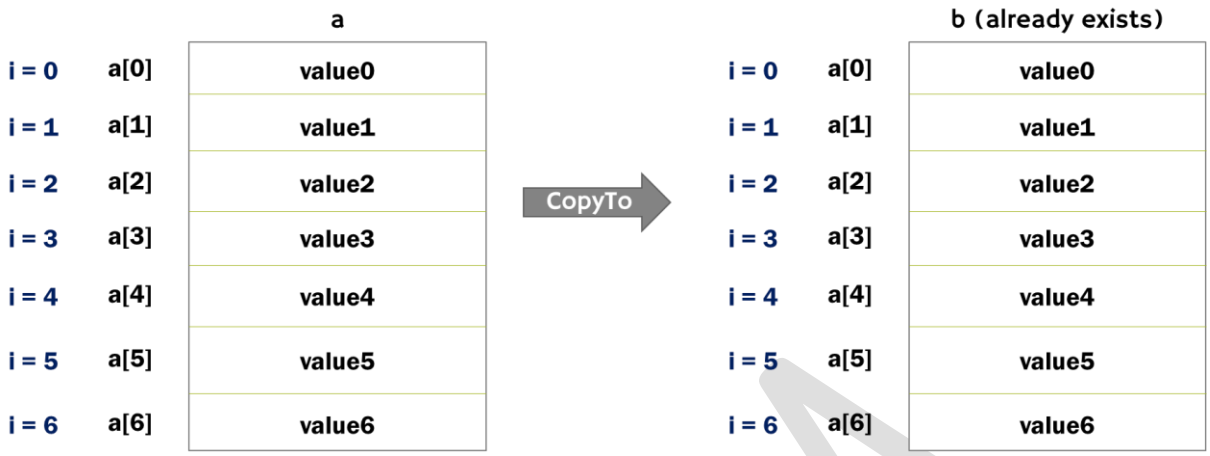
```
static void System.Reverse( System.Array array )
```

Array - Reverse( ) - Example

```
Array.Reverse(a)
```

Array – CopyTo( ) method

› This method copies all the elements from source array to destination array.



Array - CopyTo() method

```
void System.CopyTo(System.Array destinationArray, int startIndex)
```

CopyTo() - Example

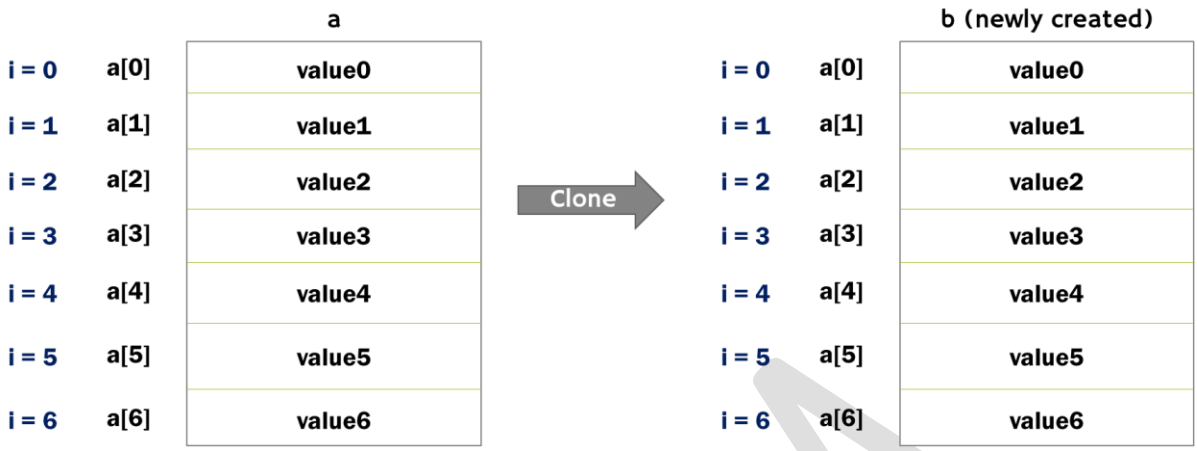
```
a.CopyTo(b, 0)
```

Parameters

- **sourceArray:** This parameter represents the array, which array you want to copy.
- **destinationArray:** This parameter represents the array, into which you want to copy the elements of source array. The destination array must exist already and should be large enough to hold new values.
- **startIndex:** This parameter represents the index of the element, from which you want to start copying.

Array – Clone( ) method

- This method copies all the elements from source array to a new destination array.



Array - Clone() method

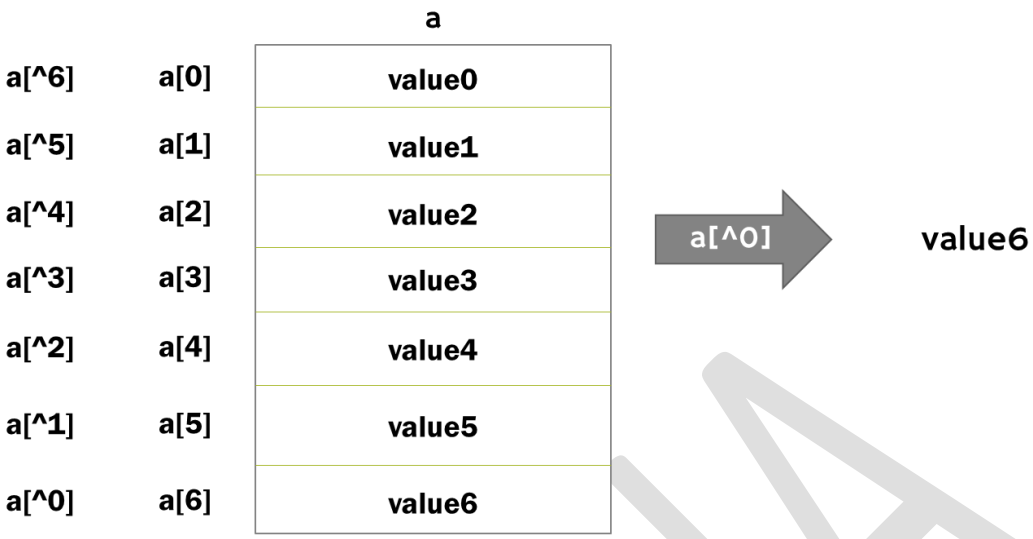
```
void System.Clone()
```

Clone() - Example

```
b = a.Clone()
```

IndexFromEnd operator

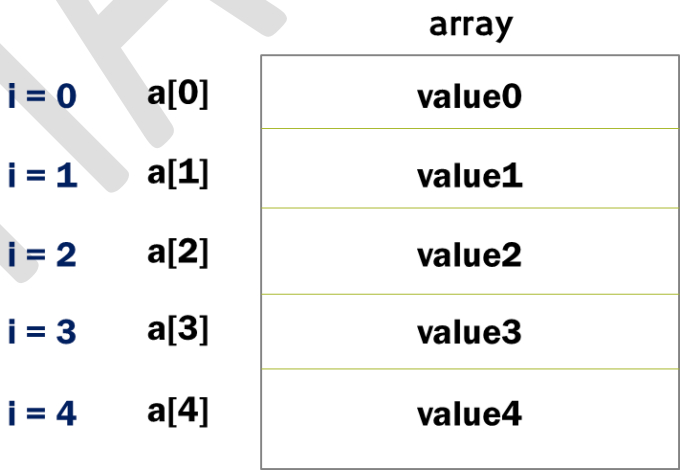
› This operator returns the index from end of the array (last element is treated as index '0').



Types of Arrays

Single-Dim Arrays

› Group of multiple rows; each row contains a single value.



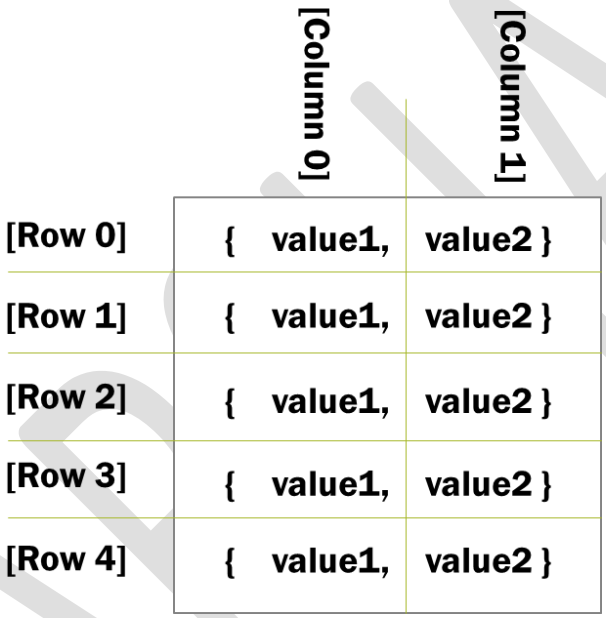
Multi-Dim Arrays

› Group of multiple rows; each row contains a group of values.

	[Column 0]	[Column 1]
[Row 0]	{ value1,	value2 }
[Row 1]	{ value1,	value2 }
[Row 2]	{ value1,	value2 }
[Row 3]	{ value1,	value2 }
[Row 4]	{ value1,	value2 }

Multi-Dim Arrays

- Stores elements in rows & columns format.
- Every row contains a series of elements.
- You can create arrays with two or dimensions, by increasing the no. of commas (,).



Multi-Dim Array

```
type[ , ]  arrayReferenceVariable = new  type[ rowSize, columnSize ];
```

Jagged Arrays

- Jagged Array is an “array of arrays”.
- The member arrays can be of any size.

Jagged Array

```
type[ ] [ ]  arrayReferenceVariable = new  type[ rowSize ] [ ];
arrayReferenceVariable[index] = new type[ size ];
```