**Arrays**

**Processing Sequences of Elements**

Table of Contents

1. Declaring and Creating Arrays
2. Accessing Array Elements
3. Console Input and Output of Arrays
4. Iterating Over Arrays Using for and foreach
5. Dynamic Arrays
   * List<T>
6. Copying Arrays

**Declaring and Creating Arrays**

* What are Arrays? - An array is a sequence of elements
  + All elements are of the same type
  + The order of the elements is fixed
  + Has fixed size (**Array.Length**)

**Declaring Arrays**

* Declaration defines the type of the elements
* Square brackets **[]** mean "array"
* Examples:
  + Declaring array of integers: int[] myIntArray;
  + Declaring array of strings: string[] myStringArray;

**Creating Arrays**

* Use the operator **new**
  + Specify array length
* Example creating (allocating) array of 5 integers: myIntArray = new int[5];

**Creating and Initializing Arrays**

* Creating and initializing can be done together: myIntArray = {1, 2, 3, 4, 5};
* The new operator is not required   
  when using curly brackets initialization

Creating Array – Example

* Creating an array that contains the names of the days of the week :

string[] daysOfWeek =

{

"Monday",

"Tuesday",

"Wednesday",

"Thursday",

"Friday",

"Saturday",

"Sunday"

};

* Days of Week - Live Demo

**Accessing Array Elements - Read and Modify Elements by Index**

How to Access Array Element?

* Array elements are accessed using the square brackets operator **[] (indexer)**
  + Array indexer takes element’s index as parameter
  + The **first** element has index **0**
  + The **last** element has index **Length-1**
* Array elements can be retrieved and changed by the [] operator

Reversing an Array – Example

* Reversing the contents of an array

int[] array = new int[] {1, 2, 3, 4, 5};

// Get array size

int length = array.Length;

// Declare and create the reversed array

int[] reversed = new int[length];

// Initialize the reversed array

for (int index = 0; index < length; index++)

{

reversed[length-index-1] = array[index];

}

* Reversing an Array - Live Demo

**Arrays: Input and Output**

**Reading Arrays From the Console**

* First, read from the console the length of the array:

int n = int.Parse(Console.ReadLine());

* Next, create the array of given size and read its elements in a **for** loop:

int[] arr = new int[n];

for (int i=0; i<n; i++)

{

arr[i] = int.Parse(Console.ReadLine());

}

Symmetry Check – Example

* Read int array from the console and check if it is symmetric:

bool isSymmetric = true;

for (int i=0; i<array.Length/2; i++)

{

if (array[i] != array[n-i-1])

{

isSymmetric = false;

}

}

* Symmetry Check - Live Demo

**Printing Arrays on the Console**

* Process all elements of the array
* Print each element to the console
* Separate elements with white space or a new line

string[] array = {"one", "two", "three"};

// Process all elements of the array

for (int index = 0; index < array.Length; index++)

{

// Print each element on a separate line

Console.WriteLine("element[{0}] = {1}",

index, array[index]);

}

* Printing Arrays - Live Demo

**Processing Array Elements Using for and foreach**

Processing Arrays: **for** Statement

* Use for loop to process an array when
  + Need to keep track of the index
  + Processing is not strictly sequential from the first to the last element
* In the loop body use the element at the loop index (array[index]):

for (int index = 0; index < array.Length; index++)

{

squares[index] = array[index] \* array[index];

}

Processing Arrays Using for Loop – Examples

* Printing array of integers in reversed order:

Console.WriteLine("Reversed: ");

for (int i = array.Length-1; i >= 0; i--)

{

Console.Write(array[i] + " ");

}

// Result: 5 4 3 2 1

* Initialize all array elements with their corresponding index number:

for (int index = 0; index < array.Length; index++)

{

array[index] = index;

}

Processing Arrays: **foreach**

* How foreach loop works? foreach (type value in array)
  + type – the type of the element
  + value – local name of variable
  + array – processing array
* Used when no indexing is needed
  + All elements are accessed one by one
  + Elements can not be modified (read only)

Processing Arrays Using foreach – Example

* Print all elements of a string[] array:

string[] capitals =

{

"Sofia",

"Washington",

"London",

"Paris"

};

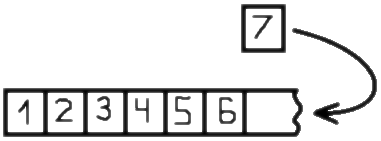
foreach (string capital in capitals)

{

Console.WriteLine(capital);

}

* Processing Arrays - Live Demo

**Resizable Arrays**

**List<T>**

**Lists (Resizable Arrays)**

* List<T> – array that can resize dynamically
  + When adding or removing elements
  + Also have indexers [] (like arrays)
  + T is the type that the list will hold
    - E.g. List<int> will hold integers
    - List<object> will hold objects
* Basic methods and properties
  + Add(T element) – adds new element to the end
  + Remove(element) – removes the element
  + Count – returns the current size of the list

List Example

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

for( int i=0; i<5; i++)

{

intList.Add(i);

}

* Is the same as:

int[] intArray = new int[5];

for( int i=0; i<5; i++)

{

intArray[i] = i;

}

* The main difference - When using lists we don't have to know the exact number of elements

**Lists vs. Arrays**

* Lets have an array with capacity of 5 elements

int[] intArray = new int[5];

* If we want to add a sixth element (we have already added 5) we have to manually resize

int[] copyArray = intArray;

int[] intArray = new int[6];

for (int i = 0; i < 5; i++)

{

intArray[i] = copyArray[i];

}

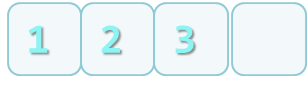
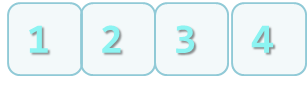
intArray[5] = newValue;

* With List<T> we simply call

list.Add(newValue);

* Lists <T> - Live Demo

**How The List<T> Works?**

* Why adding new elements is not slow?
  + When adding n elements in List<T> it resizes itself log(2)n times instead of n
* Initially a new List<T> has size of 0 elements
  + Counter for total capacity (Capacity)
  + Counter for number of used capacity (Count)
  + When created, both properties of the list have values of 0
  + When adding the first element Count becomes 1 and Capacity becomes 4
* Initially the List<T> is empty
  + When adding new element it is resized
  + But not every time - Only when it is needed
* Lets have a list with 3 elements
  + It looks like this:
  + When we add new element it is appended to the end
  + Adding a fifth element doubles the Capacity of the list
* Resizing Lists - Live Demo

**Copying Arrays**

The Array Class

* Sometimes we must copy the values from one array to another one
  + If we do it the intuitive way we would copy not only the values but the reference to the array - Changing some of the values in one array will affect the other

int[] copyArray = array;

* + The way to avoid this is using Clone() - This way only the values will be copied but not the reference

int[] copyArray = (int[])array.Clone();

Summary

* Arrays are a fixed-length sequences of elements of the same type
* Array elements are accessible by index - Can be read and modified
* Iteration over array elements can be done with for and foreach loops
* List<T> holds resizable arrays
  + Good when we don't know the number of elements initially

Free Trainings @ Telerik Academy

* + - “C# Programming @ Telerik Academy - [csharpfundamentals.telerik.com](http://csharpfundamentals.telerik.com/)
    - Telerik Software Academy - [academy.telerik.com](http://academy.telerik.com/)
    - Telerik Academy @ Facebook - [facebook.com/TelerikAcademy](http://www.facebook.com/telerikacademy)
    - Telerik Software Academy Forums - [forums.academy.telerik.com](http://forums.academy.telerik.com/)