# Advanced Programming

## Chapter 8:
## Arrays

Dr Shahriar Bijani

Spring 2016

# Reference

- **Visual C# 2012 How to Program**, Paul Deitel & Harvey Deitel, 5th Edition, Prentice Hall.

# 7.2 Arrays

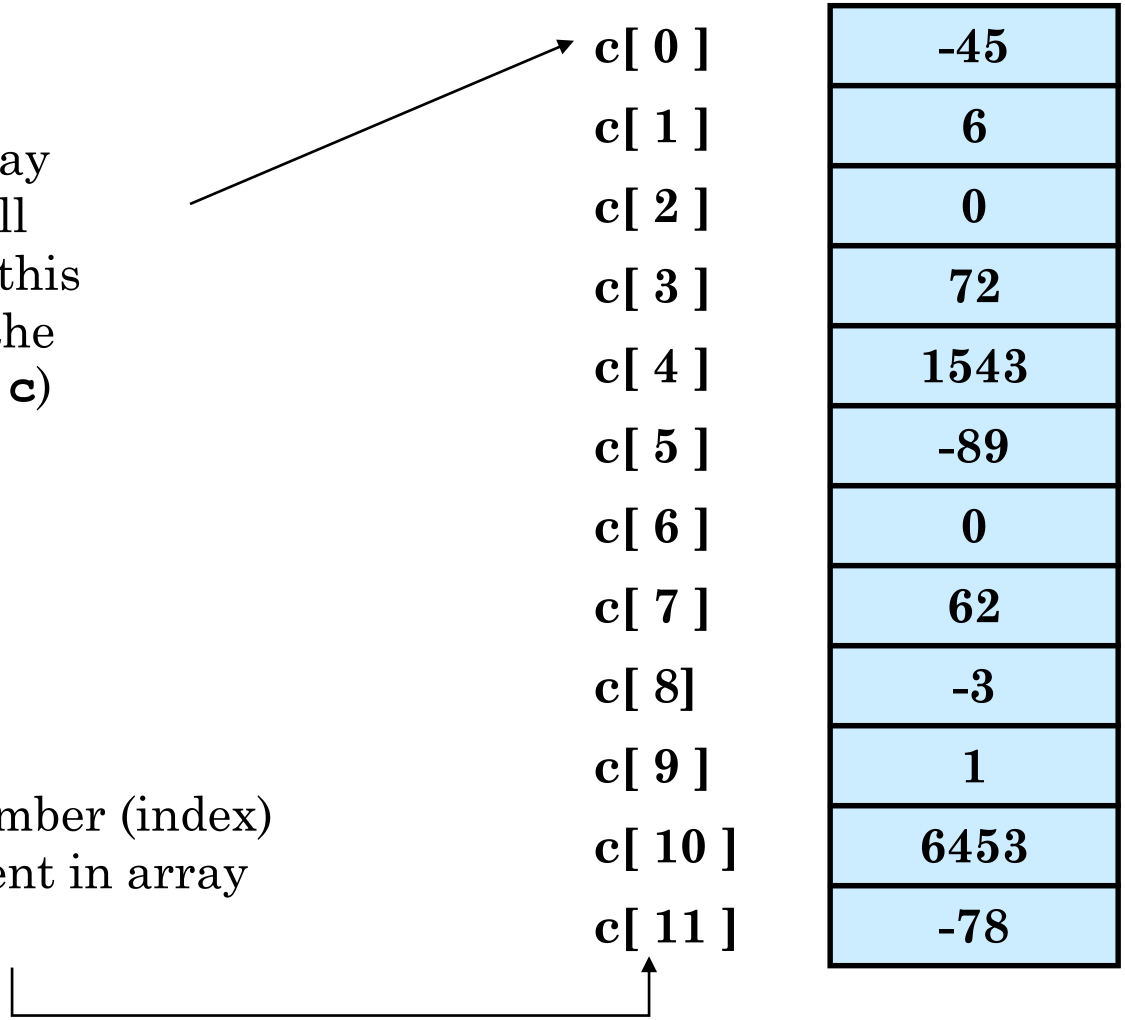
- A group of neighboring memory locations
  - Same name
  - Same type

- Refer to an element in the array by *position number* (index)

- First element is *zero*
  - First element of array **c** is **c[ 0 ]**

# 7.2 ARRAYS

Name of array (Note that all elements of this array have the same name, **c**)



| | |
|---|---|
| c[ 0 ] | -45 |
| c[ 1 ] | 6 |
| c[ 2 ] | 0 |
| c[ 3 ] | 72 |
| c[ 4 ] | 1543 |
| c[ 5 ] | -89 |
| c[ 6 ] | 0 |
| c[ 7 ] | 62 |
| c[ 8] | -3 |
| c[ 9 ] | 1 |
| c[ 10 ] | 6453 |
| c[ 11 ] | -78 |

Position number (index) of the element in array **c**

**Fig. 8.1** A 12-element array.

# 8.3 Declaring and Creating Arrays

- Each element contains one value

```
int[] c = new int[ 12 ];
```

- Array declarations and initializations can be in 2 statements:

```
int[] c;
c = new int[ 12 ];
```

- Each element of the array can be a reference to an object of that type

```
string[] s1 = new string[ 10 ],
         s2 = new string[ 50 ];
```

- Fig. 8.2

```
// Fig. 8.2: InitArray.cs     Creating an array.
using System;
public class InitArray
{
   public static void Main( string[] args )
   {
      int[] array; // declare array named array
      // create the space for array and initialize to zeros
      array = new int[ 5 ]; // 5 int elements

      Console.WriteLine( "{0}{1,8}", "Index", "Value" );

      // output each array element's value
      for ( int counter = 0; counter < array.Length; ++counter )
         Console.WriteLine( "{0,5}{1,8}",
                                   counter, array[ counter ] );
   } // end Main
} // end class InitArray
```

# Resizing & Initializing an Array

***Resizing:***

- Arrays are fixed-length entities, but you can use the static Array method Resize

```
int[] newArray = new int[ 5 ];
Array.Resize(newArray, 10 );
```

- Copies the contents of the old array into the new array
- If the new array is *smaller* than the old array, it is *cut without warning*

***Initializing:***

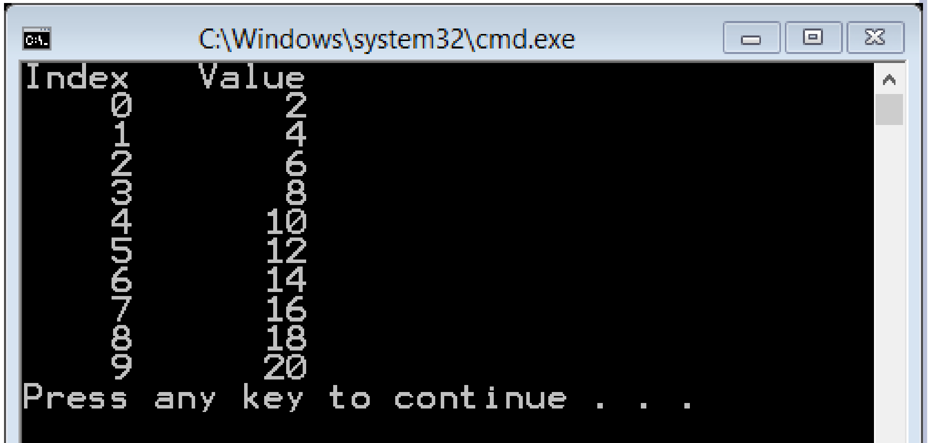
- Arrays can be initialized with *initializer lists*

```
int[] n = { 10, 20, 30, 40, 50 };
```

- number of elements in list = the size of array

# Examples of Using Arrays

```
// Fig. 8.4: InitArray.cs  Calculating values as elements of an array.
using System;
public class InitArray
{
   public static void Main( string[] args )
   {
      const int ARRAY_LENGTH = 10; // create a named constant
      int[] array = new int[ ARRAY_LENGTH ]; // create array
      // calculate value for each array element
      for ( int counter = 0; counter < array.Length; ++counter )
         array[ counter ] = 2 + 2 * counter;
      Console.WriteLine( "{0}{1,8}", "Index", "Value" ); // headings
      // output each array element's value
      for ( int counter = 0; counter < array.Length; ++counter )
         Console.WriteLine( "{0,5}{1,8}", counter, array[ counter ] );
   } // end Main
} // end class InitArray
```



```
Index   Value
    0       2
    1       4
    2       6
    3       8
    4      10
    5      12
    6      14
    7      16
    8      18
    9      20
Press any key to continue . . .
```

# Examples of Using Arrays: Fig. 8.8

```
class RollDie
{
   static void Main(){
      var randomNumbers = new Random(); // random-number generator
      var frequency = new int[7]; // array of frequency counters

      // roll die 60,000,000 times; use die value as frequency index
      for (var roll = 1; roll <= 60000000; ++roll)
         ++frequency[randomNumbers.Next(1, 7)];
      Console.WriteLine($"{"Face"}{"Frequency",10}");

      // output each array element's value
      for (var face = 1; face < frequency.Length; ++face)
           Console.WriteLine($"{face,4}{frequency[face],10}");
    }
}
```

# 8.6 FOREACH STATEMENT

- foreach iterates through all the elements of an array (or a collection)

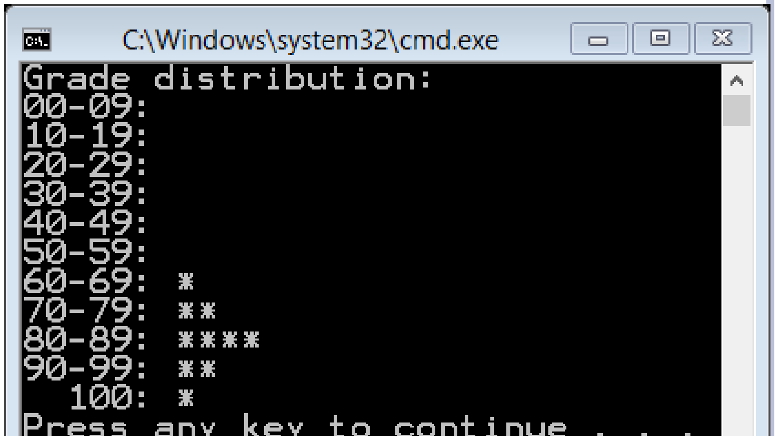
```
foreach ( type identifier in arrayName )
        statement
```

- e.g.

```
for (int counter = 0; counter < array.Length; ++counter)
          total += array[counter];

foreach ( int number in array )
         total += number;
```

- It can be used in place of for whenever you do not need access to the counter (index of array elements)

```
// Fig. 8.7: BarChart.cs
using System;
public class BarChart
{
   public static void Main( string[] args )
   {
      int[] array = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 }; // distribution
      Console.WriteLine( "Grade distribution:" );

      // for each array element, output a bar of the chart
      for ( int counter = 0; counter < array.Length; ++counter )
      {
         // output bar labels ( "00-09: ", ..., "90-99: ", "100: " )
         if ( counter == 10 )
            Console.Write( "  100: " );
         else
            Console.Write( "{0:D2}-{1:D2}: ",
               counter * 10, counter * 10 + 9 );
         // display bar of asterisks
         for ( int stars = 0; stars < array[ counter ]; ++stars )
            Console.Write( "*" );

         Console.WriteLine(); // start a new line of
      } // end outer for
   } // end Main
} // end class BarChart
```



```
Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
Press any key to continue . . .
```

# Common Programming Error 8.4

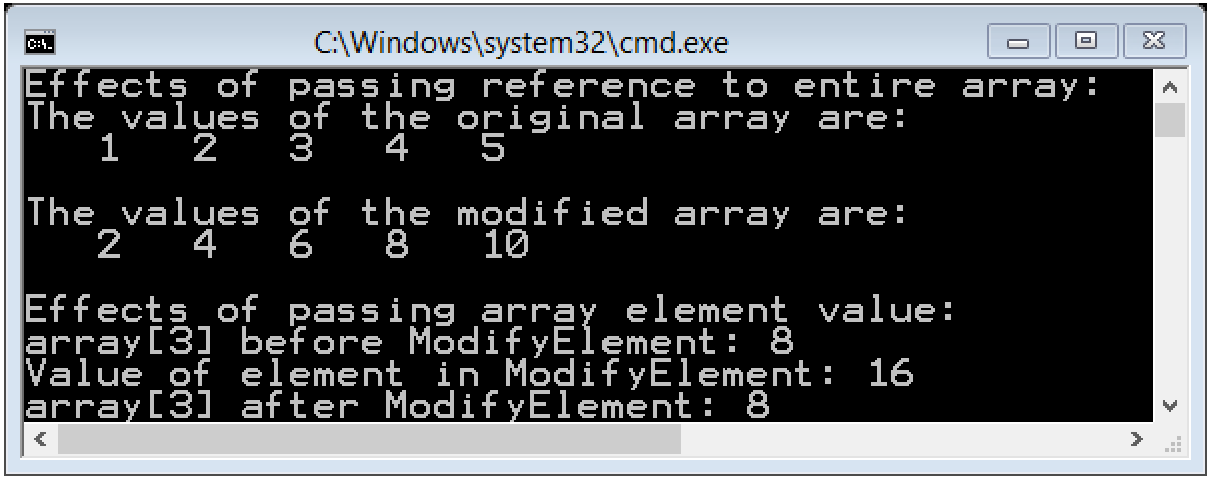
- *The foreach statement's **iteration variable** can be used **only to access array elements***
- ***It cannot be used to modify elements***.
- *Any attempt to change the value of the iteration variable in the body of a foreach statement will cause a **compilation error***.

# 8.7 Passing *Arrays / Array Elements* to Methods

- **Passing value types to methods**
  - A copy of the variable is sent
  - Any changes to variable in method do not effect the original variable
- **Passing reference types to methods**
  - A copy of the reference to the object is sent
  - Any changes to the contents of the object in the method, **do** effect the object outside the method
- Arrays are passed by reference
- Array elements are passed by value

```
// Fig. 8.13
public class PassArray
{
   // Main creates array and calls ModifyArray and ModifyElement
   public static void Main( string[] args )
   {
      int[] array = { 1, 2, 3, 4, 5 };
      Console.WriteLine("Effects of passing reference to entire array:\n" +
         "The values of the original array are:" );
      // output original array elements
      foreach ( int value in array )
         Console.Write( "   {0}", value );

      ModifyArray( array ); // pass array reference
      Console.WriteLine( "\n\nThe values of the modified array are:" );

      // output modified array elements
      foreach ( int value in array )
         Console.Write( "   {0}", value );

      Console.WriteLine("\n\nEffects of passing array element value:\n" +
         "array[3] before ModifyElement: {0}", array[ 3 ] );

      ModifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
      Console.WriteLine(
         "array[3] after ModifyElement: {0}", array[ 3 ] );
   } // end Main
```

```
 // multiply each element of an array by 2
   public static void ModifyArray( int[] array2 )
   {
      for ( int counter = 0; counter < array2.Length; ++counter )
         array2[ counter ] *= 2;
   } // end method ModifyArray

   // multiply argument by 2
   public static void ModifyElement( int element )
   {
      element *= 2;
      Console.WriteLine(
         "Value of element in ModifyElement: {0}", element );
   } // end method ModifyElement
} // end class PassArray
```



```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before ModifyElement: 8
Value of element in ModifyElement: 16
array[3] after ModifyElement: 8
```

# 8.11 Case Study

- GradeBook Using an Array (Fig 8.15-16)

# 8.4. Introduction to exception handling

```
for (var answer = 0; answer < responses.Length; ++answer)
     {
        try
        {
           ++frequency[responses[answer]];
        }
        catch (IndexOutOfRangeException ex)
        {
           Console.WriteLine(ex.Message);
           Console.WriteLine(
              $"   responses[{answer}] = {responses[answer]}\n");
        }
     }

     Console.WriteLine($"{"Rating"}{"Frequency",10}");

     // output each array element's value
     for (var rating = 1; rating < frequency.Length; ++rating)
     {
        Console.WriteLine($"{rating,6}{frequency[rating],10}");
     }
```

# 8.10 Multidimensional Arrays

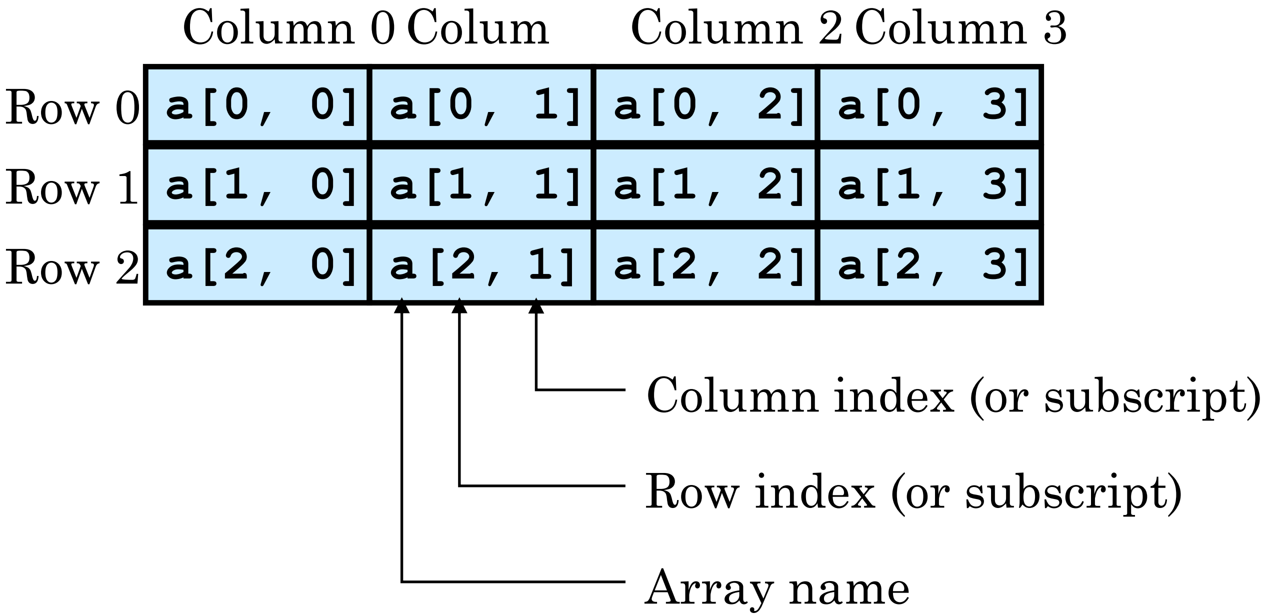
- **Multidimensional arrays:** Arrays that require 2 (or more) subscripts to identify an element

- **Types of Two-dimensional arrays:**

  - **Rectangular arrays**
    - represent tables in which each row is the same size and each column is the same size
    - first subscript identifies the element's row and the second subscript the element's column

  - **Jagged Arrays**
    - Arrays of arrays
    - Each row can be of different length

# Two-Dimentional Rectangular Arrays



|  | Column 0 | Colum | Column 2 | Column 3 |
|---|---|---|---|---|
| Row 0 | `a[0, 0]` | `a[0, 1]` | `a[0, 2]` | `a[0, 3]` |
| Row 1 | `a[1, 0]` | `a[1, 1]` | `a[1, 2]` | `a[1, 3]` |
| Row 2 | `a[2, 0]` | `a[2, 1]` | `a[2, 2]` | `a[2, 3]` |

Column index (or subscript)

Row index (or subscript)

Array name

# *Initializing Two-Dimensional Rectangular Array*

- Nested array initializing:

```
int[,] b = { { 1, 2, 3 }, { 4, 5, 6 } };
```

- Creating Two-Dimensional Arrays

```
  int[,] b = new int[2, 3];
```

- Number of initializers in each row must be the same

```
int[,] b = { { 1, 2 , 3}, { 4, 5 } };// COMPILATION ERROR
```

# *Initializing Two-Dimensional Jagged Array*

- Nested array initializing:

```
int[][] jagged = {  new int[] { 1, 2 },
                    new int[] { 3 },
                    new int[] { 4, 5, 6 } };
```

- Jagged array can not be defined in one expression:

```
int[][] c = new int[ 2 ][ 5 ]; // COMPILATION ERROR
```

- Each one-dimensional array in the jagged array must be initialized separately

```
int[][] c;
c = new int[2][]; // create 2 rows
c[0] = new int[5]; // create 5 columns for row 0
c[1] = new int[3]; // create 3 columns for row
Fig. 8.19
```

# 8.11 Case Study

- GradeBook Using a Rectangular Array (Fig 8.20)

# 8.12 Variable-Length Argument Lists

- Allow you to create methods that receive an arbitrary number of arguments.

- the keyword ***params*** and one-dimensional array-type

- *The* ***params*** *modifier can be used <u>only with the last parameter of the parameter list</u>*

```
// Fig. 8.22: Using variable-length argument lists.
public class ParamArrayTest
{  // calculate average
   public static double Average( params double[] numbers )
   {
      double total = 0.0; // initialize total
      // calculate total using the foreach statement
      foreach ( double d in numbers )
         total += d;
      return total / numbers.Length;
   } // end method Average
   public static void Main( string[] args )
   {
      double d1 = 10.0, d2 = 20.0, d3 = 30.0, d4 = 40.0;

      Console.WriteLine("d1 = {0:F1}\nd2 = {1:F1}\nd3 = {2:F1}\nd4 = {3:F1}\n",d1, d2, d3, d4 );

      Console.WriteLine( "Average of d1 and d2 is {0:F1}", Average( d1, d2 ) );
      Console.WriteLine( "Average of d1, d2 and d3 is {0:F1}", Average( d1, d2, d3 ) );
      Console.WriteLine( "Average of d1, d2, d3 and d4 is {0:F1}", Average( d1, d2, d3, d4 ) );
   } // end Main
} // end class ParamArrayTest
```

# 8.13 Using Command-Line Arguments

- It is possible to pass arguments from the **command line** to an app
- By including a parameter of type string[] in the parameter list of Main.
- By convention, this parameter is named **args**!

```
// D:\cmdArg\InitArray\InitArray\bin\Debug
using System;
public class InitArray
{
   public static void Main( string[] args ){
     if ( args.Length != 3 ) // check number of command-line arguments
         Console.WriteLine( "Error: Please re-enter the entire command, including\n" +
            "an array size, initial value and increment." );
      else {
         int arrayLength = Convert.ToInt32( args[ 0 ] ); // get array size from 1st command-line arg
         int[] array = new int[ arrayLength ]; // create array

         // get initial value and increment from command-line argument
         int initialValue = Convert.ToInt32( args[ 1 ] );
         int increment = Convert.ToInt32( args[ 2 ] );

         // calculate value for each array element
         for ( int counter = 0; counter < array.Length; ++counter )
            array[ counter ] = initialValue + increment * counter;
         Console.WriteLine( "{0}{1,8}", "Index", "Value" );

         // display array index and value
         for ( int counter = 0; counter < array.Length; ++counter )
            Console.WriteLine( "{0,5}{1,8}", counter, array[ counter ] );
      } // end else
   } // end Main
} // end class InitArray
```