

# 8

## Arrays

© 2006 Pearson Education, Inc. All rights reserved.

## 8.7 Passing Arrays and Array Elements to Methods

- To pass array argument to a method
  - Arrays are always passed by reference
  - Specify array name without brackets
    - Array `hourlyTemperatures` is declared as  
`double[] hourlyTemperatures = new double[ 24 ];`
  - The method call  
`ModifyArray( hourlyTemperatures );`
  - The method header  
`void ModifyArray( double[] b )`

© 2006 Pearson Education, Inc. All rights reserved.

```
1 // Fig. 8.13: PassArray.cs
2 // Passing arrays and individual array elements to methods.
3 using System;
4
5 public class PassArray
6 {
7     // Main creates array and calls ModifyArray and ModifyElement
8     public static void Main( string[] args )
9     {
10         int[] array = { 1, 2, 3, 4, 5 };
11
12         Console.WriteLine(
13             "Effects of passing reference to entire array:\n" +
14             "The values of the original array are: " );
15
16         // output original array elements
17         foreach ( int value in array )
18             Console.WriteLine( " {0}", value );
19
20         ModifyArray( array ); // pass array reference
21         Console.WriteLine( "\n\nThe values of the modified array are: " );
22     }
23 }
```

© 2006 Pearson Education, Inc. All rights reserved.

```
22 // output modified array elements
23 foreach ( int value in array )
24     Console.WriteLine( " {0}", value );
25
26
27 Console.WriteLine(
28     "\n\nEffects of passing array element value:\n" +
29     "array[3] before ModifyElement: {0}", array[ 3 ] );
30
31 ModifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
32 Console.WriteLine(
33     "array[3] after ModifyElement: {0}", array[ 3 ] );
34 } // end Main
35
36 // multiply each element of an array by 2
37 public static void ModifyArray( int[] array2 )
38 {
39     for ( int counter = 0; counter < array2.Length; counter++ )
40         array2[ counter ] *= 2;
41 } // end method ModifyArray
```

© 2006 Pearson Education, Inc. All rights reserved.

```
42
43 // multiply argument by 2
44 public static void ModifyElement( int element )
45 {
46     element *= 2;
47     Console.WriteLine(
48         "Value of element in ModifyElement: {0}", element );
49 } // end method ModifyElement
50 } // end class PassArray
```

© 2006 Pearson Education, Inc. All rights reserved.

## Performance Tip 8.1

Passing arrays and other objects by reference makes sense for performance reasons. If arrays were passed by value, a copy of each element would be passed. For large, frequently passed arrays, this would waste time and would consume considerable storage for the copies of the arrays—both of these problems cause poor performance.

© 2006 Pearson Education, Inc. All rights reserved.

## Software Engineering Observation 8.2

In C#, objects (including arrays) are passed by reference by default. So, a called method receiving a reference to an object in a caller can change the caller's object.

© 2006 Pearson Education, Inc. All rights reserved.

## 8.12 Variable-Length Argument Lists

### Variable-length argument lists

One-dimensional array-type argument preceded by the keyword `params` indicates that the method receives a variable number of arguments with the type of the array's elements

`params` modifier can occur only in the last entry of parameter list

Array whose elements are all the same type

© 2006 Pearson Education, Inc. All rights reserved.

```
1 // Fig. 8.22: VarargsTest.cs
2 // Using variable-length argument lists.
3 using System;
4
5 public class VarargsTest
6 {
7     // calculate average
8     public static double Average( params double[] numbers )
9     {
10         double total = 0.0; // Initialize total
11
12         // calculate total using the foreach statement
13         foreach ( double d in numbers )
14             total += d;
15
16         return total / numbers.Length;
17     } // end method Average
18 }
```

**Outline**

- Method Average receives a variable length sequence of doubles
- Calculate the total of the doubles in the array
- Access numbers.Length to obtain the size of the numbers array

VarargsTest.cs (1 of 2)

© 2006 Pearson Education, Inc. All rights reserved.

```
19 public static void Main( string[] args )
20 {
21     double d1 = 10.0;
22     double d2 = 20.0;
23     double d3 = 30.0;
24     double d4 = 40.0;
25
26     Console.WriteLine(
27         "d1 = {0:F1}\nd2 = {1:F1}\nd3 = {2:F1}\nd4 = {3:F1}\n",
28         d1, d2, d3, d4 );
29
30     Console.WriteLine( "Average of d1 and d2 is {0:F1}",
31         Average( d1, d2 ) );
32     Console.WriteLine( "Average of d1, d2 and d3 is {0:F1}",
33         Average( d1, d2, d3 ) );
34     Console.WriteLine( "Average of d1, d2, d3 and d4 is {0:F1}",
35         Average( d1, d2, d3, d4 ) );
36 } // end Main
37 } // end class VarargsTest
```

**Outline**

- Invoke method Average with two arguments
- Invoke method Average with three arguments
- Invoke method Average with four arguments

VarargsTest.cs (2 of 2)

```
d1 = 10.0
d2 = 20.0
d3 = 30.0
d4 = 40.0

Average of d1 and d2 is 15.0
Average of d1, d2 and d3 is 20.0
Average of d1, d2, d3 and d4 is 25.0
```

© 2006 Pearson Education, Inc. All rights reserved.

## Using Command-Line Arguments

### Command-line arguments

Pass arguments from the command line  
`string args[]`

Appear after the executable file name in Command Prompt

run a b

Number of arguments passed in from command line  
`args.Length`

First command-line argument  
`args[ 0 ]`

© 2006 Pearson Education, Inc. All rights reserved.

```
1 // Fig. 8.23: InitArray.cs
2 // Using command-line arguments to initialize an array.
3 using System;
4
5 public class InitArray
6 {
7     public static void Main( string[] args )
8     {
9         // check number of command-line arguments
10        if ( args.Length != 3 )
11            Console.WriteLine(
12                "Error: Please re-enter the entire command, including\n" +
13                "an array size, initial value and increment. ");
14        else
15        {
16            // get array size from first command-line argument
17            int arrayLength = Convert.ToInt32( args[ 0 ] );
18            int[] array = new int[ arrayLength ]; // create array
19
20            // get initial value and increment from command-line arguments
21            int initialValue = Convert.ToInt32( args[ 1 ] );
22            int increment = Convert.ToInt32( args[ 2 ] );
23
24            // calculate value for each array element
25            for ( int counter = 0; counter < array.Length; counter++ )
26                array[ counter ] = initialValue + increment * counter;
27
28            Console.WriteLine( "(0){1,8}", "Index", "Value" );
29        }
30    }
31 }
```

**Outline**

- Array args stores command-line arguments
- Check number of arguments passed in from the command line
- Obtain first command-line argument
- Obtain second and third command-line arguments
- Calculate the value for each array element based on command-line arguments

InitArray.cs

© 2006 Pearson Education, Inc. All rights reserved.

```

30 // display array index and value
31 for ( int counter = 0; counter < array.Length; counter++ )
32     Console.WriteLine( "(0,5)(1,8)", counter, array[ counter ] );
33 } // end else
34 } // end Main
35 } // end class Ini tArray

```

Outline

Ini tArray.cs

(2 of 3)

C:\Examp les\ch08\fl g08\_21>Ini tArray.exe  
Error: Please re-enter the entire command, including an array size, initial value and increment.

Missing command-line arguments

© 2006 Pearson Education, Inc. All rights reserved.

```

C:\Examp les\ch08\fl g08_21>Ini tArray.exe 5 0 4
Index: Val ue
0 0
1 4
2 8
3 12
4 16

```

Outline

Ini tArray.cs

(3 of 3)

```

C:\Examp les\ch08\fl g08_21>Ini tArray.exe 10 1 2
Index: Val ue
0 1
1 3
2 5
3 7
4 9
5 11
6 13
7 15
8 17
9 19

```

Three command-line arguments are 5, 0 and 4

Three command-line arguments are 10, 1 and 2

© 2006 Pearson Education, Inc. All rights reserved.

## 8.10 Multidimensional Arrays

- Multidimensional arrays
  - Two-dimensional array: Table of values consisting of rows and columns
    - Rectangular Arrays (Fig. 8.17)
      - M-by-N array
      - Often represents tables
      - Declaring two-dimensional array `b[2, 2]` with nested array initializer:

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

        - 1 and 2 initialize `b[0, 0]` and `b[0, 1]`
        - 3 and 4 initialize `b[1, 0]` and `b[1, 1]`

© 2006 Pearson Education, Inc. All rights reserved.

	Column 0	Column 1	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  
Row index  
Array name

Fig. 8.17 | Rectangular array with three rows and four columns.

© 2006 Pearson Education, Inc. All rights reserved.

## 8.10 Multidimensional Arrays (Cont.)

- Creating two-dimensional arrays with array-creation expressions
  - Can be created dynamically
    - Rectangular 3-by-4 array

```
int b[ , ];
b = new int[ 3, 4 ];
```

© 2006 Pearson Education, Inc. All rights reserved.

```

1 // Fig. 8.19: Ini tArray.cs
2 // Initial izing rectangular and jagged arrays.
3 using System;
4
5 public class Ini tArray
6 {
7     // create and output rectangular and jagged arrays
8     public static void Main( string[] args )
9     {
10        // with rectangular arrays,
11        // every column must be the same length.
12        int[ , ] rectangular = { { 1, 2, 3 }, { 4, 5, 6 } };
13        OutputArray( rectangular ); // displays array rectangular by row
14        Console.WriteLine(); // output a blank line
15
16    } // end Main

```

© 2006 Pearson Education, Inc. All rights reserved.

```
25
26 // output rows and columns of a rectangular array
27 public static void OutputArray( int[,] array )
28 {
29     Console.WriteLine( "Values in the rectangular array by row are" );
30
31     // loop through array's rows
32     for ( int row = 0; row < array.GetLength( 0 ); row++ )
33     {
34         // loop through columns of current row
35         for ( int column = 0; column < array.GetLength( 1 ); column++ )
36             Console.WriteLine( "{0}", array[ row, column ] );
37
38         Console.WriteLine(); // start new line of output
39     } // end outer for
40 } // end method OutputArray
```

© 2006 Pearson Education, Inc. All rights reserved.

## 8.10 Multidimensional Arrays (Cont.)

- Common multidimensional-array manipulations performed with **for** statements

- Many common array manipulations use **for** statements

E.g.,

```
int total = 0
for ( int row = 0; row < a.GetLength(0); row++ )
{
    for ( int column = 0; column < a.GetLength(1); column++ )
        total += a[ row, column ];
}
```