Arrays – Walkthrough

Arrays

Arrays are data structures consisting of data items of the same type

Arrays are “static” entities, in that they remain the same size once they are created

An array is a group of contiguous memory locations that all have the same name and type

To refer to a particular location or element in the array, we specify the name of the array and the position number (a value that indicates a specific location within the array) of the element to which we refer

The figure below shows an integer array called c

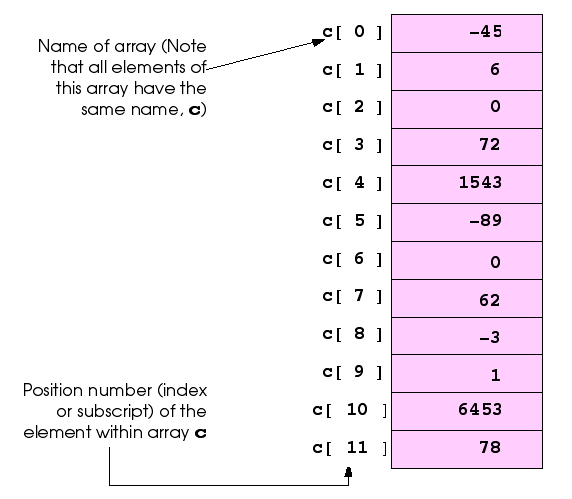
This array contains 12 elements

A program can refer to any element of an array by giving the name of the array followed by the position number of the element in square brackets ([])

The first element in every array is the zeroth element

Thus, the first element of array c is referred to as c[ 0 ], the second element of array c is referred to as c[ 1 ], the seventh element of array c is referred to as c[ 6 ] and so on

The ith element of array c is referred to as c[ i - 1 ]

Array names in C# follow the same conventions as other variable names

A 12-element array

The position number in square brackets is more formally called a **subscript** (or an index)

A subscript must be an integer or an integer expression

If a program uses an expression as a subscript, the program evaluates the expression first to determine the subscript

For example, if variable **a** is equal to **5** and variable **b** is equal to **6**, then the statement

**c[ a + b ] += 2;**

adds 2 to array element c[ 11 ]

Note that a subscripted array name is an lvalue - it can be used on the left side of an assignment to place a new value into an array element

Let us examine array c more closely

The name of the array is c

Every array in C# “knows” its own length

The length of the array is determined by the expression:

**c.Length**

The array’s 12 elements are referred to as c[ 0 ], c[ 1 ], c[ 2 ], …, c[ 11 ]

The value of c[ 0 ] is -45, the value of c[ 1 ] is 6, the value of c[ 2 ] is 0, the value of c[ 7 ] is 62 and the value of c[ 11 ] is 78

To calculate the sum of the values contained in the first three elements of array c and to store the result in variable sum, we would write

**sum = c[ 0 ] + c[ 1 ] + c[ 2 ];**

To divide the value of the seventh element of array c by 2 and assign the result to the variable x, we would write

**x = c[ 6 ] / 2;**

The brackets that enclose the subscript of an array are operators

Brackets have the same level of precedence as parentheses

The table below shows precedence and associativity of C#’s operators

They are displayed top to bottom in decreasing order of precedence, with their associativity and type

Note that the **++** and **- -** operators in the first row represent the postincrement and postdecrement operators, while the **++** and **- -** operators in the second row represent the preincrement and predecrement operators

Also, notice that in the first row the associativity is mixed

This is because the associativity of the postincrement and postdecrement operators is right to left, while the associativity for the other operators is left to right

|  |  |  |
| --- | --- | --- |
| **Operators** | **Associativity** | **Type** |
| () [] . ++ - - | left to right | highest (unary postfix) |
| ++ - - + - ! (type) | right to left | unary (unary prefix) |
| \* / % | left to right | multiplicative |
| + - | left to right | additive |
| < <= > >= | left to right | relational |
| == != | left to right | equality |
| & | left to right | logical AND |
| ^ | left to right | logical exclusive OR |
| | | left to right | logical inclusive OR |
| && | left to right | conditional AND |
| || | left to right | conditional OR |
| ?: | right to left | conditional |
| = += -= \*= /= %= | right to left | assignment |

Summary of precedence and associativity operators

Declaring and Allocating Arrays

Arrays occupy space in memory

The programmer specifies the type of the elements and uses operator **new** to allocate dynamically the number of elements required by each array

Arrays are allocated with new because arrays are objects and all objects must be created with new

We will see an exception to this rule shortly

The declaration

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

allocates 12 elements for integer array c

The preceding statement can also be performed in two steps as follows:

**int[] c; // declares the array**

**c = new int[ 12 ]; // allocates the reference to the array**

When arrays are allocated, the elements are initialized to zero for the numeric primitive-data-type variables, to false for bool variables and to null for reference types

Memory may be reserved for several arrays with a single declaration

The following declaration reserves 100 elements for string array b and 27 elements for string array x

**string[] b = new string[ 100 ], x = new string[ 27 ];**

Similarly, the following declaration reserves 10 elements for array1 and 20 elements for array2 (both of type double):

**double[] array1 = new double[ 10 ], array2 = new double[ 20 ];**

Arrays may be declared to contain most data types

In an array of value types, every element of the array contains one value of the declared type

For example, every element of an int array is an int value

In an array of reference types, every element of the array is a reference to an object of the data type of the array

For example, every element of a string array is a reference to a string

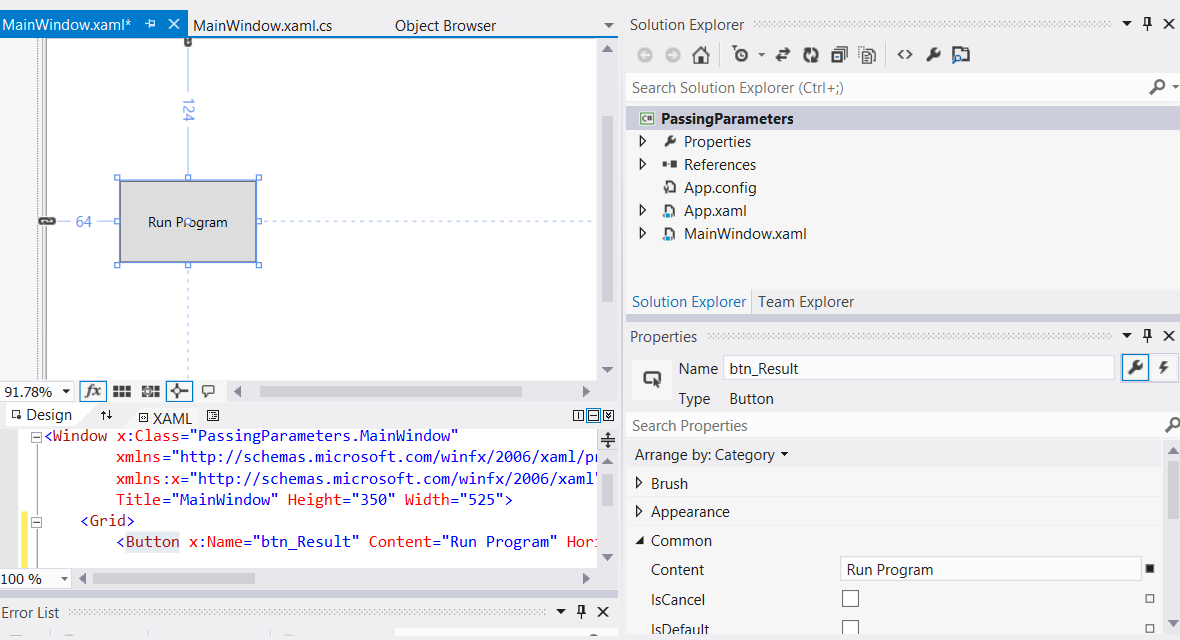
Each of these string references has the value null by default

We are going to build a basic application which has a single button, which when clicked will:

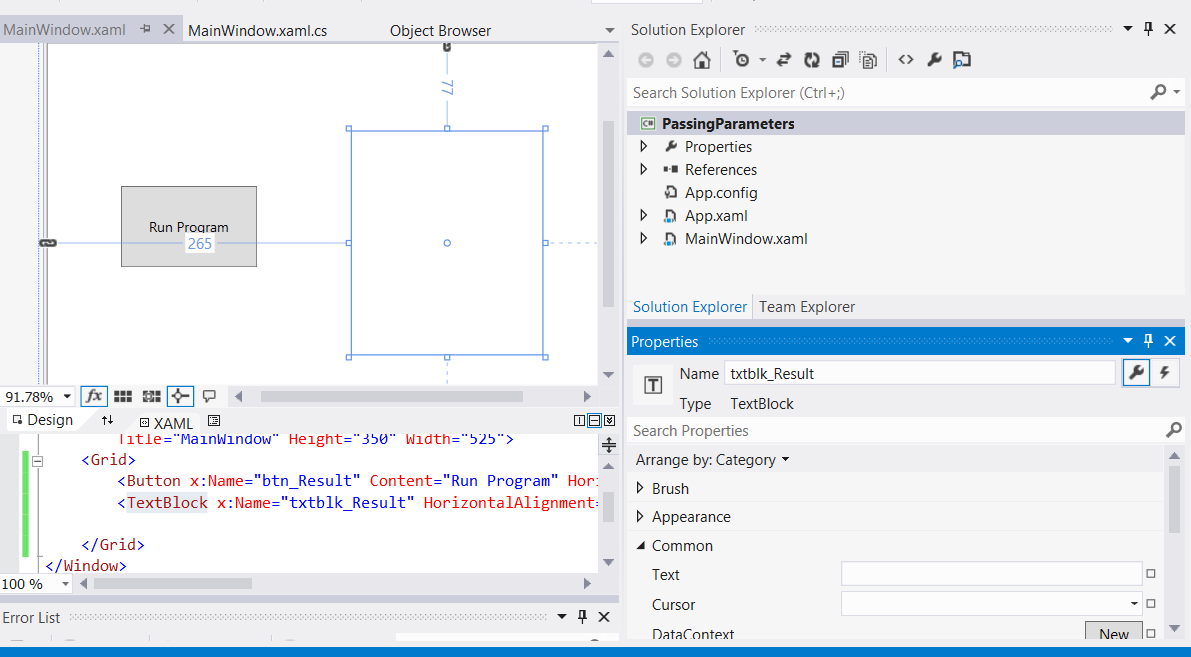
* Create and initialise 3 arrays
* Display the index and contents of each arrays elements

The aim of this tutorial is to show how to create and initialise arras.

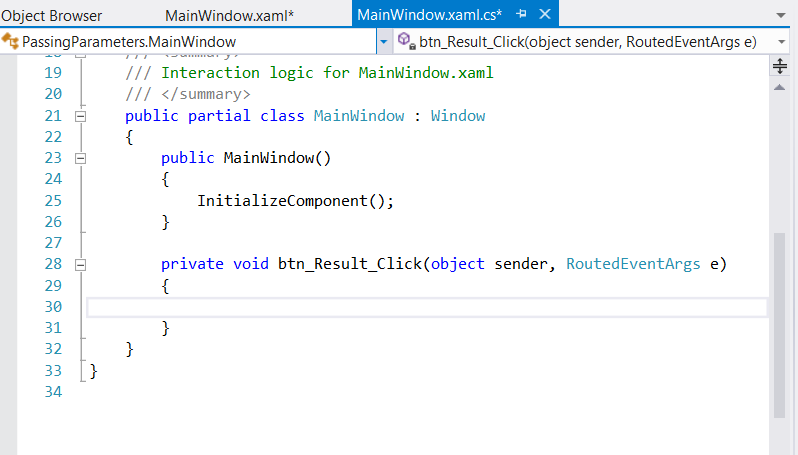
1. Create a new WPF project
2. Drag across a button from the toolbox
   1. set the “Name” property to btn\_Result
   2. set the “Content” property to “Run Program”



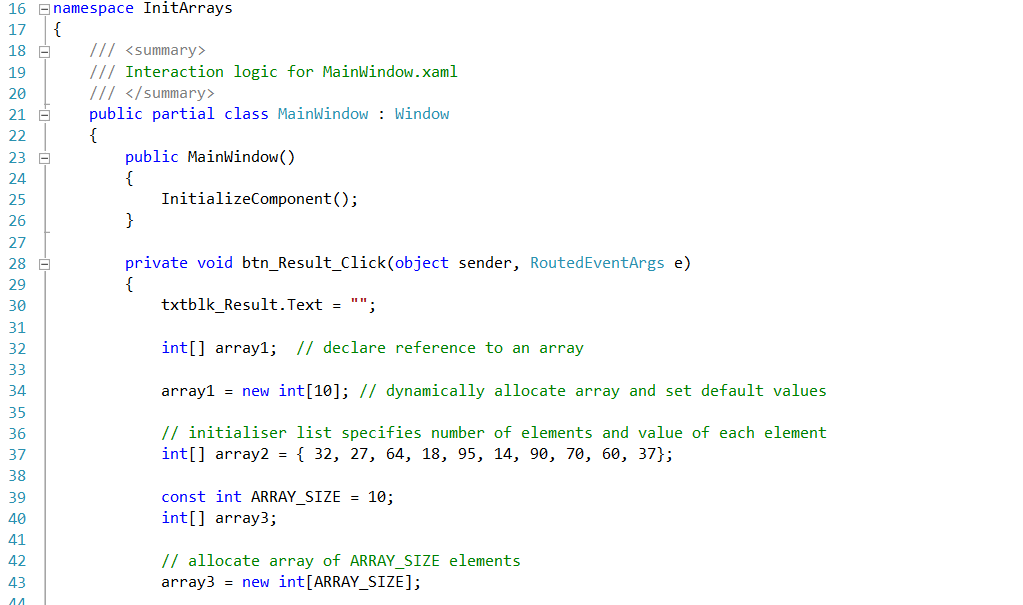
1. Drag across a text block and from the toolbox
   1. set the “Name” property to txtblk\_Result
   2. set the “Text” property to blank

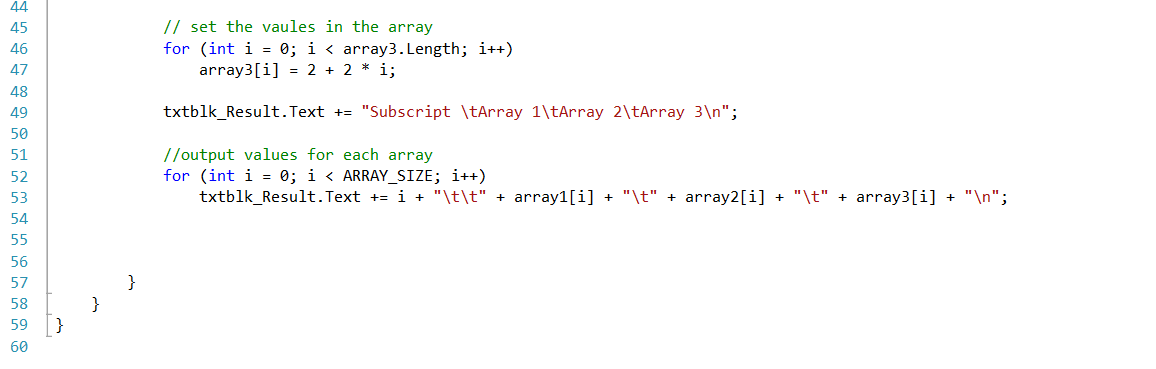


1. Double-click on the “Run Program” button to automatically create an event handler in the MainWindow.xaml.cs file, which is a method called btn\_Result\_Click



1. We are now going to write the code for this event handler. The final code can be viewed in the screenshot below followed by a full explanation





Allocating an Array and Initializing Its Elements - code walkthrough

Line 32 declares arary1 as a reference to an array of integers

Each element in the array is of type int

The variable array1 is of type int[], which denotes an array whose elements are of type int

Line 34 allocates the 10 elements of the array with new and assigns the array to reference array1

Each element of this array has the default value 0

Line 37 creates another int array and initializes each element using an initializer list

In this case, the number of elements in the initializer list determines the array’s size

For example, line 37 creates a 10-element array with the indices 0–9 and the values 32, 27, 64, and so on

Note that this declaration does not require the new operator to create the array object—the compiler allocates memory for the object when it encounters an array declaration that includes an initializer list

On line 39, we create constant integer ARRAY\_SIZE using keyword const

A constant must be initialized in the same statement where it is declared and cannot be modified thereafter

If an attempt is made to modify a constant after it is declared, the compiler issues a syntax error

Constants also are called named constants

They often are used to make a program more readable and are usually denoted with variable names in all capital letters

On line 43, we create integer array array3 of length 10 using the ARRAY\_SIZE named constant

The for structure in lines 46–47 initializes each element in array array3

The values are generated by multiplying each successive value of the loop counter by 2 and adding 2 to the product

After this initialization, array array3 contains the even integers 2, 4, 6, …, 20

The for structure in lines 52–53 uses the values in arrays array1, array2 and array3 to display in the text block.

Zero-based counting (remember, array subscripts start at 0) allows the loop to access every element of the array

The constant ARRAY\_SIZE in the for structure condition (line 52) specifies the arrays’ lengths