

Contents

Single-Dimensional Arrays	1
Introduction	1
Example	2
Abridged Syntaxes	2

Single-Dimensional Arrays

Introduction

You can define a single-dimensional array as follow:

```
<type>[] arrayName;
```

where

- <type> can be any data-type and specifies the data-type of the array elements.
- arrayName is an identifier that you will use to access and modify the array elements.

Before using an array, you must specify the number of elements in the array as follows:

```
arrayName = new <type>[<number of elements>];
```

where <type> is a type as before, and <number of elements>, called the *size declarator*, is a strictly positive integer which will correspond to the size of the array.

- An element of a single-dimensional array can be accessed and modified by using the name of the array and the index of the element as follows:

```
// Assigns <value> to the <index> element of the array  
↪ arrayName.
```

```
arrayName[<index>] = <value>;
```

```
// Display the <index> element of the array arrayName.  
Console.WriteLine(arrayName[<index>]);
```

The index of the first element in an array is always zero; the index of the second element is one, and the index of the last element is the size of the array minus one. As a consequence, if you specify an index greater or equal to the number of elements, a run-time error will happen.

Indexing starting from 0 may seem surprising and counter-intuitive, but this is a largely respected convention across programming languages and

computer scientists. Some insights on the reasons behind this (collective) choice can be found in this answer on Computer Science Educators.

Example

In the following example, we define an array named `myArray` with three elements of type integer, and assign 10 to the first element, 20 to the second element, and 30 to the last element.

```
int[] myArray;  
myArray = new int[3]; // 3 is the size declarator  
// We can now store 3 ints in this array,  
// at index 0, 1 and 2  
  
myArray[0] = 10; // 0 is the subscript, or index  
myArray[1] = 20;  
myArray[2] = 30;
```

If we were to try to store a fourth value in our array, at index 3, using e.g.

```
myArray[3] = 40;
```

our program would compile just fine, which may seem surprising. However, when executing this program, *array bounds checking* would be performed and detect that there is a mismatch between the size of the array and the index we are trying to use, resulting in a quite explicit error message:

```
Unhandled Exception: System.IndexOutOfRangeException:  
  ↳ Index was outside the bounds of the array at  
  ↳ Program.Main()
```

Abridged Syntaxes

If you know the number of elements when you are defining an array, you can combine declaration and assignment on one line as follows:

```
<type>[] arrayName = new <type>[<number of elements>];
```

So, we can combine the first two lines of the previous example and write:

```
int[] myArray = new int[3];
```

We can even initialize *and* give values on one line:

```
int[] myArray = new int[3] { 10, 20, 30 };
```

And that statement can be rewritten as any of the following:

```
int[] myArray = new int[] { 10, 20, 30 };  
int[] myArray = new[] { 10, 20, 30 };  
int[] myArray = { 10, 20, 30 };
```

But, we should be careful, the following would cause an error:

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

If we use the shorter notation, we *have to* give the values at initialization, we cannot re-use this notation once the array has been created.

Other datatypes, and even objects, can be stored in arrays in a perfectly similar way:

```
string[] myArray = { "Bob", "Mom", "Train", "Console" };
```

```
// Assume there is a class called Rectangle.
```

```
Rectangle[] arrayOfRectangle = new Rectangle[5];
```