Contents

Simple Loops and Length

Custom Size and Loops

One of the benefits of arrays is that they allow you to specify the number of their elements at run-time: the size declarator can be a variable, not just an integer literal. Hence, depending on run-time conditions such as user input, we can have enough space to store and process any number of values.

In order to access the elements of whose size is not known until run-time, we will need to use a loop. If the size of `myArray` comes from user input, it wouldn't be safe to try to access a specific element like `myArray[5]`, because we cannot guarantee that the array will have at least 6 elements. Instead, we can write a loop that uses a counter variable to access the array, and use the loop condition to ensure that the variable does not exceed the size of the array.

Example

In the following example, we get the number of elements at run-time from the user, create an array with the appropriate size, and fill the array.

```
Console.WriteLine("What is the size of the array that you
 ↪  want?");
int size = int.Parse(Console.ReadLine());
int[] customArray = new int[size];

int counter = 0;
while (counter < size)
{
    Console.WriteLine($"Enter the {counter + 1}th value");
    customArray[counter] = int.Parse(Console.ReadLine());
    counter++;
}
```

Observe that:

- If the user enters a negative value or a string that does not correspond to an integer for the `size` value, our program will crash: we are not performing any user-input validation here, to keep our example compact.
- The loop condition is `counter < size` because we do *not* want the loop to execute when `counter` is equal to `size`. The last valid index in `customArray` is `size - 1`.
- We are asking for the `{counter +1}th` value because we prefer not to confuse the user by asking for the "0th" value. Note that a more sophisticated program would replace "th" with "st", "nd" and "rd" for the first three values.

The Length Property

Every single-dimensional array has a property called `Length` that returns the number of the elements in the array (or size of the array).

To process an array whose size is not fixed at compile-time, we can use this property to find out the number of elements in the array.

Example

```
int counter2 = 0;
while (counter2 < customArray.Length)
{
    Console.WriteLine($"{counter2}:
 ↪  {customArray[counter2]}.");
    counter2++;
}
```

Observe that this code does not need the variable `size`.

Note: You *cannot* use the length property to change the size of the array, that is, entering

```
int[] test = new int[10];
test.Length = 9;
```

would return, at compile time,

```
Compilation error (line 8, col 3): Property or indexer
 ↪  'System.Array.Length' cannot be assigned to --it is
 ↪  read only.
```

When a field is marked as 'read only,' it means the attribute can only be initialized during the declaration or in the constructor of a class. We receive this error because the array attribute, 'Length,' can not be changed once the array is already declared. Resizing arrays will be discussed in the section: Changing the Size.

For Loops With Arrays

- Previously, we learned that you can iterate over the elements of an array using a **while** loop. We can also process arrays using **for** loops, and in many cases they are more concise than the equivalent **while** loop.
- For example, consider this code that finds the average of all the elements in an array:

```
int[] homeworkGrades = {89, 72, 88, 80, 91};
int counter = 0;
int sum = 0;
while(counter < 5)
{
    sum += homeworkGrades[counter];
    counter++
}
double average = sum / 5.0;
```

- This can also be written with a **for** loop:

```
int sum = 0;
for(int i = 0; i < 5; i++)
{
    sum += homeworkGrades[i];
}
double average = sum / 5.0;
```

- In a **for** loop that iterates over an array, the counter variable is also used as the array index
- Since we did not need to use the counter variable outside the body of the loop, we can declare it in the loop header and limit its scope to the loop's body
- Using a **for** loop to access array elements makes it easy to process "the whole array" when the size of the array is user-provided:

```
Console.WriteLine("How many grades are there?");
int numGrades = int.Parse(Console.ReadLine());
int[] homeworkGrades = new int[numGrades];
for(int i = 0; i < numGrades; i++)
{
    Console.WriteLine($"Enter grade for homework {i+1}");
    homeworkGrades[i] = int.Parse(Console.ReadLine());
}
```

- You can use the `Length` property of an array to write a loop condition, even if you did not store the size of the array in a variable. For

example, this code does not need the variable `numGrades`:

```
int sum = 0;
for(int i = 0; i < homeworkGrades.Length; i++)
{
    sum += homeworkGrades[i];
}
double average = (double) sum / homeworkGrades.Length;
```

- In general, as long as the loop condition is in the format `i < <arrayName>.Length` (or, equivalently, `i <= <arrayName>.Length - 1`), the loop will access each element of the array.