CHAPTER 9

Foreach and Iterator

This chapter will discuss the foreach statement and iterators block. The foreach statement is used to iterate over arrays or objects that implement the `System.Collections.IEnumerable` or `System.Collections.Generic.IEnumerable<T>` interfaces. The iterators block is also used to iterate through an array or object collection based on the state machine generated automatically by the C# compiler.

State Machine

Throughout this chapter, you will explore how the C# compiler generates the state machine for an iterator block, what the states are that the state machine contains, and how the state transition takes place in the iterator sections. Therefore, it's important to begin this chapter with a brief overview of state machines.

The C# compiler automatically generates a state machine for the iterator code block when an iterator method is defined in a program. A state machine is a model by which an object alters its behavior when its internal state changes in response to events. A state is a unique condition in which a state machine does some specific action in its lifetime. An event in context of the state machine is something that triggers the state machine to do a transition, and a transition demonstrates the actions when a state machine receives an event depending on its current state. An action refers to what a state machine performs during a transition.

In .NET, the C# compiler generates the state machine for the iterator block used in a program and maintains its state on each of the transitions of the state based on an event (called the `MoveNext` method, which we will explore later in this chapter). For each of the transitions in the state machine, the CLR does some action, for example, process the iterated item that was defined in the respective state and return it to the caller.

Foreach Statement

The foreach statement iterates over each of the elements in a collection and associated statements will process each of the elements it retrieves from the collection on iteration. The following section explains how to declare a foreach statement in a program, and later you will explore how the C# compiler compiles the foreach statement used in a program and also how the CLR handles it in runtime.
Declaration

The foreach statement declared in the program will have the following syntax:

```csharp
foreach ( local_variable_type(LT) iteration_variable(IV) in an_expression(E) )
{
    Statement or Statements (S)
}
```

The local variable type (LT) defines the type of the iteration variable (IV) but if var is used for the LT, it is said to be an implicitly typed iteration variable and its type is the element type of the Enumerator object that we get from the expression E. The iteration variable can't be updated from the Statement or Statements (S) used for the foreach block or a compile-time error occurs, as shown below, while trying to update the iteration variable number from the statement block of the foreach statement, as shown in Listing 9-1.

```
Error 19 Cannot assign to 'number' because it is a 'foreach iteration variable'
```

Listing 9-1. An Example of a Foreach Statement

```csharp
using System;
using System.Collections.Generic;

namespace Ch09
{
    class Program
    {
        static void Main(string[] args)
        {
            List<int> numbers = new List<int>
            {
                1,2,3,4,4,6,7,8,9,10
            };
            foreach (int number in numbers)
            {
                Console.Write("{0}\t", number);
            }
        }
    }
}
```

The program in Listing 9-1 produces the following output:

```
1       2       3       4       4       6       7       8       9       10
```

Internal of the Foreach

In the following sections, we will explore how the C# compiler treats a foreach statement in compile time and also how the CLR takes care of it in runtime when there is a foreach statement used in a program.