An **indexer** allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a **virtual array**. You can then access the instance of this class using the array access operator ([ ]).

Syntax

A one dimensional indexer has the following syntax −

```
element-type this[int index] {

   // The get accessor.
   get {
      // return the value specified by index
   }

   // The set accessor.
   set {
      // set the value specified by index
   }
}
```

Use of Indexers

Declaration of behavior of an indexer is to some extent similar to a property. similar to the properties, you use **get** and **set** accessors for defining an indexer. However, properties return or set a specific data member, whereas indexers returns or sets a particular value from the object instance. In other words, it breaks the instance data into smaller parts and indexes each part, gets or sets each part.

Defining a property involves providing a property name. Indexers are not defined with names, but with the **this** keyword, which refers to the object instance. The following example demonstrates the concept −

```
using System;

namespace IndexerApplication {

   class IndexedNames {
      private string[] namelist = new string[size];
      static public int size = 10;

      public IndexedNames() {
         for (int i = 0; i < size; i++)
         namelist[i] = "N. A.";
      }
      public string this[int index] {
         get {
            string tmp;

            if( index >= 0 && index <= size-1 ) {
               tmp = namelist[index];
            } else {
               tmp = "";
            }

            return ( tmp );
         }
         set {
            if( index >= 0 && index <= size-1 ) {
               namelist[index] = value;
            }
         }
      }
      static void Main(string[] args) {
         IndexedNames names = new IndexedNames();
         names[0] = "Zara";
         names[1] = "Riz";
         names[2] = "Nuha";
         names[3] = "Asif";
         names[4] = "Davinder";
         names[5] = "Sunil";
         names[6] = "Rubic";

         for ( int i = 0; i < IndexedNames.size; i++ ) {
            Console.WriteLine(names[i]);
         }
         Console.ReadKey();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
Zara
Riz
Nuha
Asif
Davinder
Sunil
Rubic
N. A.
N. A.
N. A.
```

Overloaded Indexers

Indexers can be overloaded. Indexers can also be declared with multiple parameters and each parameter may be a different type. It is not necessary that the indexes have to be integers. C# allows indexes to be of other types, for example, a string.

The following example demonstrates overloaded indexers −

```
using System;

namespace IndexerApplication {
   class IndexedNames {
      private string[] namelist = new string[size];
      static public int size = 10;

      public IndexedNames() {
         for (int i = 0; i < size; i++) {
            namelist[i] = "N. A.";
         }
      }
      public string this[int index] {
         get {
            string tmp;

            if( index >= 0 && index <= size-1 ) {
               tmp = namelist[index];
            } else {
               tmp = "";
            }

            return ( tmp );
         }
         set {
            if( index >= 0 && index <= size-1 ) {
               namelist[index] = value;
            }
         }
      }

      public int this[string name] {
         get {
            int index = 0;

            while(index < size) {
               if (namelist[index] == name) {
                return index;
               }
               index++;
            }
            return index;
         }
      }

      static void Main(string[] args) {
         IndexedNames names = new IndexedNames();
         names[0] = "Zara";
         names[1] = "Riz";
         names[2] = "Nuha";
         names[3] = "Asif";
         names[4] = "Davinder";
         names[5] = "Sunil";
         names[6] = "Rubic";

         //using the first indexer with int parameter
         for (int i = 0; i < IndexedNames.size; i++) {
            Console.WriteLine(names[i]);
         }

         //using the second indexer with the string parameter
         Console.WriteLine(names["Nuha"]);
         Console.ReadKey();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result −

```
Zara
Riz
Nuha
Asif
Davinder
Sunil
Rubic
N. A.
N. A.
N. A.
2
```