VT Academy
Thinking Ahead, Beyond Boundaries

Object Oriented Programming

Array, String, Enum and Structure

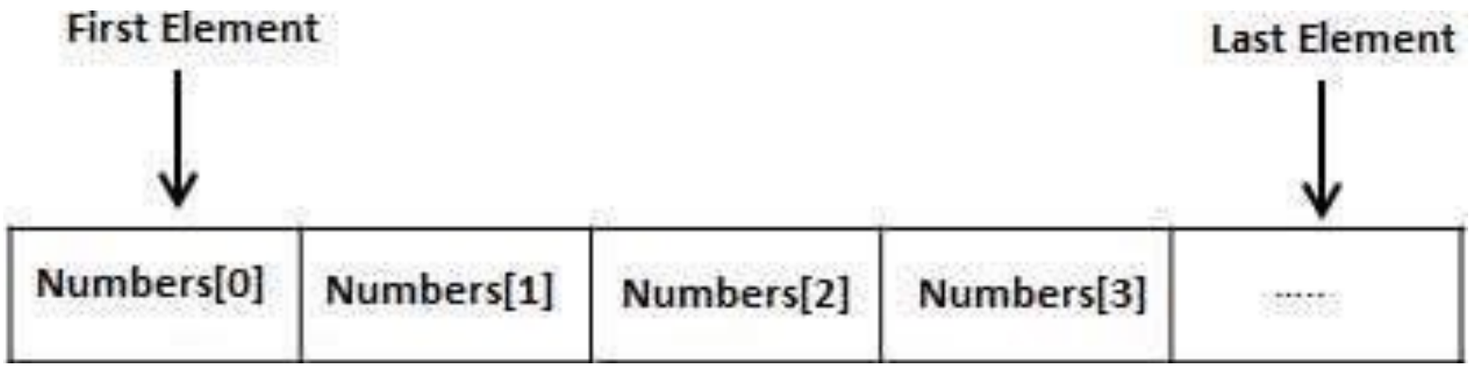
Session 3

Objectives

- Working with arrays
- Working with strings
- Enum data type
- Structure in C#

Concept of Arrays

- An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.



Declaring and Intitializing Arrays

- To declar an array, use following syntax:

```
datatype[] arrayName;
```

  where,

  - *datatype* is used to specify the type of elements in the array.
  - *[ ]* specifies the rank of the array. The rank specifies the size of the array.
  - *arrayName* specifies the name of the array.

- Initializing an array:

```
double[] balance = new double[10];
```

Assigning Values to an Array

- Assign values to individual array elements, by using the index number:

```
double[] balance = new double[10];
balance[0] = 4500.0;
```

- Assign values to the array at the time of declaration:

```
double[] balance = {2340.0, 4523.69, 3421.0};
```

- You can also create and initialize an array:

```
int[] marks = new int[5] {99,  98, 92, 97, 95};
```

- You can copy an array variable into another target array variable

```
int [] marks = new int[] {99,  98, 92, 97, 95};
int[] score = marks;
```

- Accessing an element in array:

```
double salary = balance[9];
```

Array Example

```
using System;
namespace ArrayDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         int[]  n = new int[10]; // n is an array of 10 integers
         int i,j;
         // initialize elements of array n
         for ( i = 0; i < 10; i++ )
         {
            n[ i ] = i + 100;
         }

         // output each array element's value
         for (j = 0; j < 10; j++ )
         {
            Console.WriteLine("Element[{0}] = {1}", j, n[j]);
         }
         Console.ReadKey();
      }
   }
}
```

String in C#

- In C#, you can use strings as array of characters. However, more common practice is to use the **string** keyword to declare a string variable. The string keyword is an alias for the **System.String** class.
- Creating a String Object
  - By assigning a string literal to a String variable
  - By using a String class constructor
  - By using the string concatenation operator (+)
  - By retrieving a property or calling a method that returns a string
  - By calling a formatting method to convert a value or an object to its string representation

String Example

```
using System;
namespace StringDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            // from string literal and string concatenation
            string fname, lname;
            fname = "Bill";
            lname = "Gates";
            string fullname = fname + lname;
            Console.WriteLine("Full Name: {0}", fullname);
            // by using string constructor
            char[] letters = { 'H', 'e', 'l', 'l','o' };
            string greetings = new string(letters);
            Console.WriteLine("Greetings: {0}", greetings);
            //methods returning string
            string[] sarray = { "Hello", "From", "VTC", "Academy" };
            string message = String.Join(" ", sarray);
            Console.WriteLine("Message: {0}", message);
            //formatting method to convert a value
            DateTime waiting = new DateTime(2017, 10, 10, 17, 58, 1);
            string chat = String.Format("Message sent at {0:t} on {0:D}", waiting);
            Console.WriteLine("Message: {0}", chat);
        }
    }
}
```

Properties of the String Class

VTC Academy
Thinking Ahead, Beyond Boundaries

Sr.No	Property
1	**Chars** Gets the *Char* object at a specified position in the current *String* object.
2	**Length** Gets the number of characters in the current String object.

Popular String Manipulation Methods

VTC Academy
Thinking Ahead, Beyond Boundaries

Sr.No	Methods
1	**public static int Compare(string strA, string strB)** Compares two specified string objects and returns an integer that indicates their relative position in the sort order.
2	**public static string Concat(string str0, string str1)** Concatenates two string objects.
3	**public bool Equals(string value)** Determines whether the current String object and the specified String object have the same value.
4	**public static string Join(string separator, params string[] value)** Concatenates all the elements of a string array, using the specified separator between each element.
5	**public string Replace(char oldChar, char newChar)** Replaces all occurrences of a specified Unicode character in the current string object with the specified Unicode character and returns the new string.
6	**public string[] Split(char[] separator)** Returns a string array that contains the substrings in the current string object, delimited by elements of a specified Unicode character array.
7	**public string Substring(int count)** Returns a substring from this instance. The substring starts at a specified character position and continues to the end of the string.
8	**public string Trim()** Removes all leading and trailing white-space characters from the current String object.

Comparing Strings

The following example demonstrates comparing 2 strings:

```
using System;
namespace ComaprestringDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            string str1 = "This is test string";
            string str2 = "This is text string";

            if (String.Compare(str1, str2) == 0)
            {
                Console.WriteLine(str1 + " and " + str2 +  " are equal.");
            }
            else
            {
                Console.WriteLine(str1 + " and " + str2 + " are not equal.");
            }
            Console.ReadKey() ;
        }
    }
}
```

Check String Contains String

The following example demonstrates check string contains string:

```
using System;
namespace ContaintringDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            string str = "This is test";
            if (str.Contains("test"))
            {
                Console.WriteLine("The string /"test/" was found.");
            }
            Console.ReadKey() ;
        }
    }
}
```

Getting a Substring

The following example demonstrates getting the substring:

```
using System;
namespace SubStringDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            string str = "Last night I dreamt of San Pedro";
            Console.WriteLine(str);
            string substr = str.Substring(20);
            Console.WriteLine(substr);
        }
    }
}
```

Joining Strings

The following example demonstrates joining strings:

```
using System;
namespace JoinStringDemo
{
   class Program
   {
      static void Main(string[] args)
      {
         string[] strarray = new string[] {"Down the way nights are dark",
         "And the sun shines daily on the mountain top",
         "I took a trip on a sailing ship",
         "And when I reached Jamaica",
         "I made a stop"};

         string str = String.Join("\n", strarray);
         Console.WriteLine(str);
      }
   }
}
```

Enum Data Type

- An enumeration is a set of named integer constants. An enumerated type is declared using the **enum** keyword.
- C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.
- Syntax:

```
enum <enum_name>
{
   enumeration list
};
```

- Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it. By default, the value of the first enumeration symbol is 0.

```
enum Days {Sun, Mon, Tue, Wed, Thu, Fri, Sat};
```

Enum Data Type

```
using System;
namespace EnumDemo
{
    class Program
    {
        enum Days {Sun, Mon, Tue, Wed, Thu, Fri, Sat};

        static void Main(string[] args)
        {
            int WeekdayStart = (int)Days.Mon;
            int WeekdayEnd = (int)Days.Fri;
            Console.WriteLine("Monday: {0}", WeekdayStart);
            Console.WriteLine("Friday: {0}", WeekdayEnd);
            Console.ReadKey();
        }
    }
}
```

Introduction to Structure

- In C#, a structure is a value type data type. It helps you to make a single variable hold related data of various data types. The **struct** keyword is used for creating a structure.

- Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book:

  - Title
  - Author
  - Subject
  - Book ID

Defining a Structure

- To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member for your program.
- For example, here is the way you can declare the Book structure:

```
struct Books
{
   public string title;
   public string author;
   public string subject;
   public int book_id;
};
```

Structure Example

VTC Academy
Thinking Ahead, Beyond Boundaries

```
using System;
struct Books {
   public string title;
   public string author;
   public int bookid;
};
public class testStructure {
   public static void Main(string[] args) {
      Books Book1;   // Declare Book1 of type Book
      Books Book2;   // Declare Book2 of type Book
      // book 1 specification
      Book1.title = "C Programming";
      Book1.author = "Steve Jobs";
      Book1.bookid = 6495407;
      // book 2 specification
      Book2.title = "Telecom Billing";
      Book2.author = "Bill Gates";
      Book2.bookid = 6495700;
      // print Book1 info
      Console.WriteLine("Book 1 title : {0}", Book1.title);
      Console.WriteLine("Book 1 author : {0}", Book1.author);
      Console.WriteLine("Book 1 bookid :{0}", Book1.bookid);
      // print Book2 info
      Console.WriteLine("Book 2 title : {0}", Book2.title);
      Console.WriteLine("Book 2 author : {0}", Book2.author);
      Console.WriteLine("Book 2 bookid : {0}", Book2.bookid);
      Console.ReadKey();
   }
}
```

Features of C# Structures

- Structures can have methods, fields, indexers, properties, operator methods, and events
- Structures can have defined constructors, but not destructors. However, you cannot define a default constructor for a structure. The default constructor is automatically defined and cannot be changed.
- Unlike classes, structures cannot inherit other structures or classes
- Structures cannot be used as a base for other structures or classes.
- A structure can implement one or more interfaces
- Structure members cannot be specified as abstract, virtual, or protected

Summary

- An array stores a fixed-size sequential collection of elements of the same type
- In C#, you can use strings as array of characters by using **string** keyword
- An enumeration is a set of named integer constants. An enumerated type is declared using the **enum** keyword.
- In C#, a structure is a value type data type. It helps you to make a single variable hold related data of various data types. The **struct** keyword is used for creating a structure.
- Structures can have methods, fields, indexers, properties, operator methods, and events