Lecture No 7 (Types of Function)

Muhammad Siddique

Functions

- A method is a group of statements that together perform a task.
- Every C# program has at least one class with a method named Main.
  - To use a method, you need to
    - Define the method
    - Call the method
- General format of function is mentioned below:

```
<Access Specifier> <Return Type> <Method Name>(Parameter List)
{
   Method Body
}
```

Functions Parameters

Access Specifier:

- This determines the visibility of a variable or a method from another class.

Return Type:

- A method may return a value. The return type is the data type of the value the method returns.
- If the method is not returning any values, then the return type is void.

Method Name:

- Method name is a unique identifier and it is case sensitive.
- It cannot be same as any other identifier declared in the class.

Functions Parameters

Parameter List:

- Enclosed between parentheses, the parameters are used to pass and receive data from a method.
- The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.

Method Body:

- This contains the set of instructions needed to complete the required activity.

Define Functions with Parameters (Example)

- This code shows a function "FindMax" that takes two integer values and returns the larger of the two.
- It has public access specifier, so it can be accessed from outside the class using an instance of the class.

```
class NumberManipulator {

   public int FindMax(int num1, int num2) {
      /* local variable declaration */
      int result;

      if (num1 > num2)
         result = num1;
      else
         result = num2;

      return result;
   }
   ...
}
```

Calling a Functions (By same class object)

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public int FindMax(int num1, int num2) {
         /* local variable declaration */
         int result;

         if (num1 > num2)
            result = num1;
         else
            result = num2;
         return result;
      }

      static void Main(string[] args) {
         /* local variable definition */
         int a = 100;
         int b = 200;
         int ret;
         NumberManipulator n = new NumberManipulator();

         //calling the FindMax method
         ret = n.FindMax(a, b);
         Console.WriteLine("Max value is : {0}", ret );
         Console.ReadLine();
      }
   }
}
```

```
Max value is : 200
```

Calling a Functions (In other class)

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public int FindMax(int num1, int num2) {
         /* local variable declaration */
         int result;

         if(num1 > num2)
            result = num1;
         else
            result = num2;

         return result;
      }
   }
   class Test {
      static void Main(string[] args) {
         /* local variable definition */
         int a = 100;
         int b = 200;
         int ret;
         NumberManipulator n = new NumberManipulator();

         //calling the FindMax method
         ret = n.FindMax(a, b);
         Console.WriteLine("Max value is : {0}", ret );
         Console.ReadLine();
      }
   }
}
```

```
Max value is : 200
```

Calling a Functions (Recursive call)

- A method can call itself. This is known as recursion.

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public int factorial(int num) {
         /* local variable declaration */
         int result;
         if (num == 1) {
            return 1;
         } else {
            result = factorial(num - 1) * num;
            return result;
         }
      }
      static void Main(string[] args) {
         NumberManipulator n = new NumberManipulator();
         //calling the factorial method {0}", n.factorial(6));
         Console.WriteLine("Factorial of 7 is : {0}", n.factorial(7));
         Console.WriteLine("Factorial of 8 is : {0}", n.factorial(8));
         Console.ReadLine();
      }
   }
}
```

```
Factorial of 6 is: 720
Factorial of 7 is: 5040
Factorial of 8 is: 40320
```

Passing Parameters to a Function

- There are three ways that parameters can be passed to a method which are as follows
  - Value parameters.
  - Reference parameters.
  - Output parameters.

Passing Parameters to a Function by Value

- This is the default mechanism for passing parameters to a method.
- In this mechanism, when a method is called, a new storage location is created for each value parameter.
- The values of the actual parameters are copied into them.
- Hence, the changes made to the parameter inside the method have no effect on the argument.

Passing Parameters to a Function by Value (Example)

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public void swap(int x, int y) {
         int temp;

         temp = x; /* save the value of x */
         x = y;    /* put y into x */
         y = temp; /* put temp into y */
      }
      static void Main(string[] args) {
         NumberManipulator n = new NumberManipulator();

         /* local variable definition */
         int a = 100;
         int b = 200;

         Console.WriteLine("Before swap, value of a : {0}", a);
         Console.WriteLine("Before swap, value of b : {0}", b);

         /* calling a function to swap the values */
         n.swap(a, b);

         Console.WriteLine("After swap, value of a : {0}", a);
         Console.WriteLine("After swap, value of b : {0}", b);

         Console.ReadLine();
      }
   }
}
```

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
After swap, value of b :200
```

Passing Parameters to a Function by Reference

- A reference parameter is a reference to a memory location of a variable.
- When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters.
- The reference parameters represent the same memory location as the actual parameters that are supplied to the method.
- You can declare the reference parameters using the ref keyword.

Passing Parameters to a Function by Reference (Example)

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public void swap(ref int x, ref int y) {
         int temp;

         temp = x; /* save the value of x */
         x = y;    /* put y into x */
         y = temp; /* put temp into y */
      }
      static void Main(string[] args) {
         NumberManipulator n = new NumberManipulator();

         /* local variable definition */
         int a = 100;
         int b = 200;

         Console.WriteLine("Before swap, value of a : {0}", a);
         Console.WriteLine("Before swap, value of b : {0}", b);

         /* calling a function to swap the values */
         n.swap(ref a, ref b);

         Console.WriteLine("After swap, value of a : {0}", a);
         Console.WriteLine("After swap, value of b : {0}", b);

         Console.ReadLine();
      }
   }
}
```

```
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 200
After swap, value of b : 100
```

Passing Parameters to a Function by Output

- A return statement can be used for returning only one value from a function.
- However, using output parameters, you can return two values from a function.
- Output parameters are similar to reference parameters, except that they transfer data out of the method rather than into it.

Passing Parameters to a Function by Output (Example)

```
using System;

namespace CalculatorApplication {
   class NumberManipulator {
      public void getValue(out int x ) {
         int temp = 5;
         x = temp;
      }
      static void Main(string[] args) {
         NumberManipulator n = new NumberManipulator();

         /* local variable definition */
         int a = 100;

         Console.WriteLine("Before method call, value of a : {0}", a);

         /* calling a function to get the value */
         n.getValue(out a);

         Console.WriteLine("After method call, value of a : {0}", a);
         Console.ReadLine();
      }
   }
}
```

```
Before method call, value of a : 100
After method call, value of a : 5
```

Comparison b/w Reference vs Output Function

- A reference parameter is a reference to a memory location of a variable.
- Both ref and out parameter treated same at compile-time but different at run-time.

Sr #	**Reference Method**	**Output Method**
1.	It is necessary the parameters should initialize before it pass to ref.	It is not necessary to initialize parameters before it pass to out.
2.	It is not necessary to initialize the value of a parameter before returning to the calling method.	It is necessary to initialize the value of a parameter before returning to the calling method.
3.	The passing of value through ref parameter is useful when the called method also need to change the value of passed parameter.	The declaring of parameter through out parameter is useful when a method return multiple values.
4.	When ref keyword is used the data may pass in bi-directional.	When out keyword is used the data only passed in unidirectional.