VB.NET - SUB PROCEDURES

As we mentioned in the previous chapter, Sub procedures are procedures that do not return any value. We have been using the Sub procedure Main in all our examples. We have been writing console applications so far in these tutorials. When these applications start, the control goes to the Main Sub procedure, and it in turn, runs any other statements constituting the body of the program.

Defining Sub Procedures

The **Sub** statement is used to declare the name, parameter and the body of a sub procedure. The syntax for the Sub statement is:

```
[Modifiers] Sub SubName [(ParameterList)]
    [Statements]
End Sub
```

Where,

- ***Modifiers***: specify the access level of the procedure; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- ***SubName***: indicates the name of the Sub
- ***ParameterList***: specifies the list of the parameters

Example

The following example demonstrates a Sub procedure *CalculatePay* that takes two parameters *hours* and *wages* and displays the total pay of an employee:

```
Module mysub
   Sub CalculatePay(ByVal hours As Double, ByVal wage As Decimal)
      'local variable declaration
      Dim pay As Double
      pay = hours * wage
      Console.WriteLine("Total Pay: {0:C}", pay)
   End Sub
   Sub Main()
      'calling the CalculatePay Sub Procedure
      CalculatePay(25, 10)
      CalculatePay(40, 20)
      CalculatePay(30, 27.5)
      Console.ReadLine()
   End Sub
End Module
```

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

```
Total Pay: $250.00
Total Pay: $800.00
Total Pay: $825.00
```

Passing Parameters 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. So, the changes made to the parameter inside the method have no effect on the argument.

In VB.Net, you declare the reference parameters using the **ByVal** keyword. The following example

demonstrates the concept:

```
Module paramByval
   Sub swap(ByVal x As Integer, ByVal y As Integer)
      Dim temp As Integer
      temp = x ' save the value of x
      x = y    ' put y into x
      y = temp 'put temp into y
   End Sub
   Sub Main()
      ' local variable definition
      Dim a As Integer = 100
      Dim b As Integer = 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 '
      swap(a, b)
      Console.WriteLine("After swap, value of a : {0}", a)
      Console.WriteLine("After swap, value of b : {0}", b)
      Console.ReadLine()
   End Sub
End Module
```

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

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

It shows that there is no change in the values though they had been changed inside the function.

Passing Parameters 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.

In VB.Net, you declare the reference parameters using the **ByRef** keyword. The following example demonstrates this:

```
Module paramByref
   Sub swap(ByRef x As Integer, ByRef y As Integer)
      Dim temp As Integer
      temp = x ' save the value of x
      x = y    ' put y into x
      y = temp 'put temp into y
   End Sub
   Sub Main()
      ' local variable definition
      Dim a As Integer = 100
      Dim b As Integer = 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 '
      swap(a, b)
      Console.WriteLine("After swap, value of a : {0}", a)
      Console.WriteLine("After swap, value of b : {0}", b)
      Console.ReadLine()
   End Sub
End Module
```

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

```
Before swap, value of a : 100
```

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