Sub and Function Procedures

Chapter 7

Introduction

- So far, most of the code has been inside a single method for an event
  - Fine for small programs, but inconvenient for large ones
  - Much better to divide program into manageable pieces
- Benefits of modularization
  - Avoids repeat code (reuse a function many times in one program)
  - Promotes software reuse (reuse a function in another program)
  - Promotes good design practices (Specify function interfaces)
  - Promotes debugging (can test an individual module to make sure it works properly)
- General procedures: procedures not associated with specific events
  - Sub
  - Function
  - Property

Sub Procedures

- The purpose of a Sub procedure is to operate and manipulate data within some specific context
- A general procedure is invoked by using its defined name
  - For example: Message()
  - You've been using Sub Procedures all the time:
    - E.g. g.DrawLine(Pens.Blue, 10, 10, 40, 40)

Creating a General Sub Procedure

- Ensure that the Code window is activated by:
  - Double clicking on a Form, or
  - Pressing the F7 function key, or
  - Selecting the Code item from the View menu
- Type a procedure declaration into the Code window
  - Public Sub procedure-name()
- Visual Basic will create the procedure stub
- Type the required code

Exchanging Data with a General Procedure

- Syntax for calling a Sub procedure into action: procedure-name(argument list)

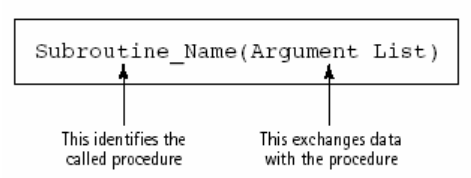


Figure 7-5: Calling a Sub Procedure

Exchanging Data with a General Procedure (continued)

- A general Sub procedure declaration must include:
  - Keyword Sub
  - Name of the general procedure
    - The rules for naming Sub procedures are the same as the rules for naming variables
  - Names of any parameters
- Parameter: the procedure's declaration of what data it will accept
- Argument: the data sent by the calling function
- Individual data types of each argument and its corresponding parameter must be the same

Exchanging Data with a General Procedure (continued)

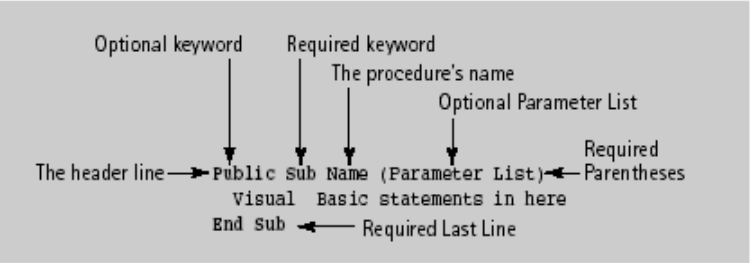


Figure 7-7: The Structure of a General Sub Procedure

Example

```
lstResult.Items.Clear()

ExplainPurpose()
lstResult.Items.Add("")

Public Sub ExplainPurpose()
    lstResult.Items.Add("This program displays a sentence")
    lstResult.Items.Add("identifying two numbers and their sum.")
End Sub
```

Code Re-Use

- If in another place in the code you wanted to explain the purpose, you can just invoke the subroutine:

```
Public Sub OtherCode(…)
        ExplainPurpose()
        ‘ Presumably other code here
End Sub
```

- Avoids duplicate the same code in many places
- If you ever want to change the code, only one place needs to be changed

Passing Parameters

- You can send items to a Sub procedure

```
Sum(2, 3)
```

```
Public Sub Sum(num1 As Double, num2 As Double)
    Console.WriteLine(num1+num2)
End Sub
```

- In the Sum Sub procedure, 2 will be stored in num1 and 3 will be stored in num2 and the sum will be output to the console

Passing Variables

- We can pass variables too:

```
x = 2
y = 3
Sum(x,y)                ‘ Same as Sum(2, 3)
```

- The variables are evaluated prior to calling the subroutine, and their values are accessible via the corresponding variable names in the sub

Population Density Sub

- Subroutine to calculate population density:

```
Public Sub CalculateDensity(ByVal state As String, _
                ByVal pop As Double, _
                ByVal area As Double)
   Dim rawDensity, density As Double
   rawDensity = pop / area
   density = Math.Round(rawDensity, 1)     ' Round to 1 decimal place
   Console.Write("The density of " & state & " is " & density)
   Console.WriteLine(" people per square mile.")
End Sub
```

Parameters and Arguments

```
CalculateDensity("Alaska", 627000, 591000)
```

Arguments – what you send to a Sub procedure

```
Public Sub CalculateDensity(ByVal state As String, _
                            ByVal pop As Double, _
                            ByVal area As Double)
```

Parameters – place holders for what the sub procedure receives

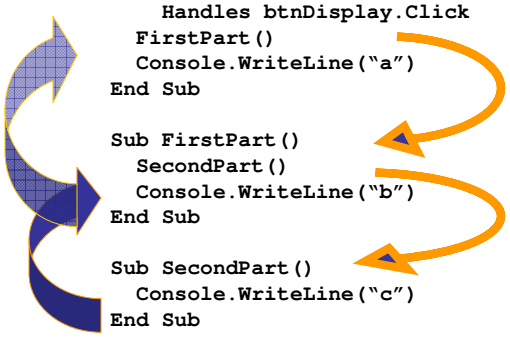
If ByVal left off, VB.NET will add it

Code Reuse

- By making CalculateDensity a procedure subroutine, we can reuse it, e.g.:

  CalculateDensity("Hawaii", 1212000, 6471)

Sub Procedures Calling Other Sub Procedures



```
Private Sub btnDisplay_Click(...)
    Handles btnDisplay.Click
  FirstPart()
  Console.WriteLine("a")
End Sub

Sub FirstPart()
  SecondPart()
  Console.WriteLine("b")
End Sub

Sub SecondPart()
  Console.WriteLine("c")
End Sub
```

Output:

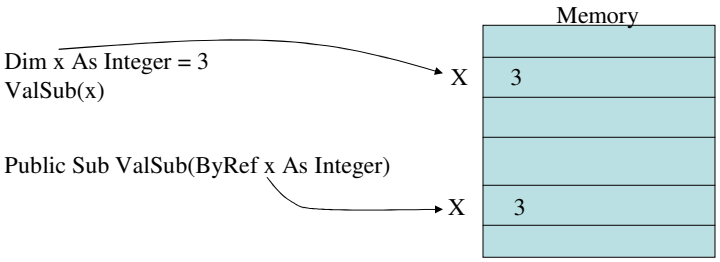
c
b
a

In Class Exercise

- Write a Sub procedure that takes as arguments an animal and sound for the "Old McDonald Had A Farm" song and outputs the verse, e.g.:
  - Old McDonald had a farm, E-I-E-I-O.
  - And on his farm he had a cow, E-I-E-I-O.
  - With a moo moo here, and a moo moo there,
  - Here a moo, there a moo, everywhere a moo moo.
  - Old McDonald had a farm, E-I-E-I-O
- Complete the program in the Form Load event to output the verses for a cow, chicken, and lamb.

Passing by Value

- ByVal stands for "By Value"
  - Default mode, VB.NET adds this for you if you leave it off
- ByVal parameters retain their original value after Sub procedure terminates
  - Can think of this as a copy of the variable is sent in



ByVal Example

```
Public Sub CallingSub()
    Dim y As Integer
    y = 5
    Console.WriteLine("y is " & y)
    ValSub(y)
    Console.WriteLine("y is " & y)
End Sub

Public Sub ValSub(ByVal x As Integer)
    x = 10
    Console.WriteLine(" x is " & x)
End Sub
```

Output?

ByVal Example – Y to X

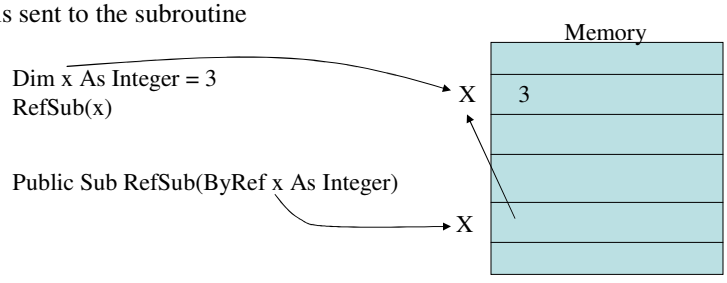
```
Public Sub CallingSub()
    Dim x As Integer
    x = 5
    Console.WriteLine("x is " & x)
    ValSub(x)
    Console.WriteLine("x is " & x)
End Sub

Public Sub ValSub(ByVal x As Integer)
    x = 10
    Console.WriteLine("x is " & x)
End Sub
```

Output?

Passing by Reference

- ByRef stands for "By Reference"
  - You can think of this as a reference, or pointer, to the original variable is sent to the subroutine



- ByRef parameters can be changed by the Sub procedure and retain the new value after the Sub procedure terminates

ByRef Example

```
Public Sub CallingSub()
    Dim y As Integer
    y = 5
    Console.WriteLine("y is " & y)
    RefSub(y)
    Console.WriteLine("y is " & y)
End Sub

Public Sub RefSub(ByRef x As Integer)
    x = 10
    Console.WriteLine(" x is " & x)
End Sub
```

Output?

ByVal Example – Y to X

```
Sub CallingSub()
    Dim x As Integer
    x = 5
    Console.WriteLine("x is " & x)
    RefSub(x)
    Console.WriteLine("x is " & x)
End Sub

Sub RefSub(ByRef x As Integer)
    x = 10
    Console.WriteLine("x is " & x)
End Sub
```

Any Difference in Output?

Local Variables

- Variables declared inside a Sub procedure with a Dim statement
- Parameters are also considered local variables; their values are gone when the subroutine exits (unless parameters were passed ByRef)

In-Class Exercise

- Write a subroutine that swaps two integer variables; e.g. Swap(x,y) results in exchanging the values in X and Y

Function Procedures

- A function directly returns a single value to its calling procedure
- Types of functions:
  - Intrinsic
  - User-defined

Function Procedures (continued)

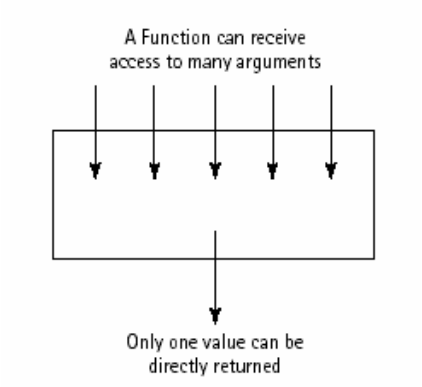


Figure 7-13: A Function Directly Returns a Single Value

Function Procedures (continued)

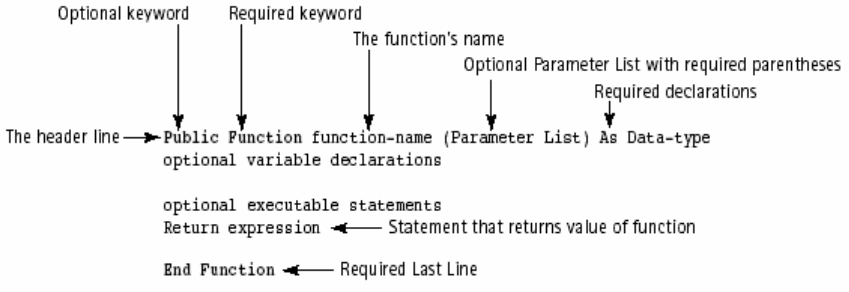


Figure 7-14: The Structure of a Function Procedure

Calling a Function Procedure

- To call a function procedure:
  - Give the function's name
  - Pass any data to it in the parentheses following the function name
- Arguments of the called function are the items enclosed within the parentheses in a calling statement

Calling a Function Procedure (continued)

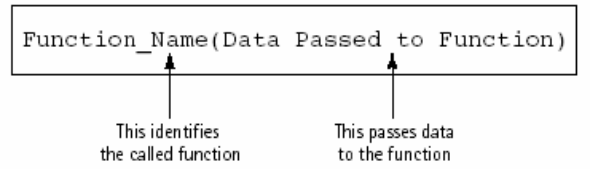


Figure 7-15: Calling and Passing Data to a Function

Sample

```
Private Sub btnDetermine_Click(...)
    Handles btnDetermine.Click
  Dim name As String
  name = txtFullName.Text
  txtFirstname.Text = FirstName(name)
End Sub

Public Function FirstName(ByVal name As String) As String
  Dim firstSpace As Integer
  firstSpace = name.IndexOf(" ")
  Return name.Substring(0, firstSpace)
End Function
```

Function call

Return statement

Having Several Parameters

```
Private Sub btnCalculate_Click(...)
    Handles btnCalculate.Click
  Dim a, b As Double
  a = CDbl(txtSideOne.Text)
  b = CDbl(txtSideTwo.Text)
  txtHyp.Text = CStr( Hypotenuse(a, b) )
End Sub

Public Function Hypotenuse( ByVal a As Double, _
                    ByVal b As Double ) As Double
  Return Math.Sqrt(a ^ 2 + b ^ 2)
End Function
```

User-Defined Functions Having No Parameters

```
Private Sub btnDisplay_Click(...) _
    Handles btnDisplay.Click
  txtBox.Text = Saying()
End Sub

Public Function Saying() As String
  Return InputBox("What is your" _
           & " favorite saying?")
End Function
```

Comparing Function Procedures with Sub Procedures

- Subs are accessed using a call statement
- Functions are called where you would expect to find a literal or expression
- For example:
  - Result = *functionCall*
  - Console.WriteLine (*functionCall*)

Functions vs. Procedures

- Both can perform similar tasks
- Both can call other subs and functions
- Use a function when you want to return one and only one value
  - A function or sub can also be declared with ByRef arguments to return multiple values back through the argument list

Collapsing a Procedure with a Region Directive

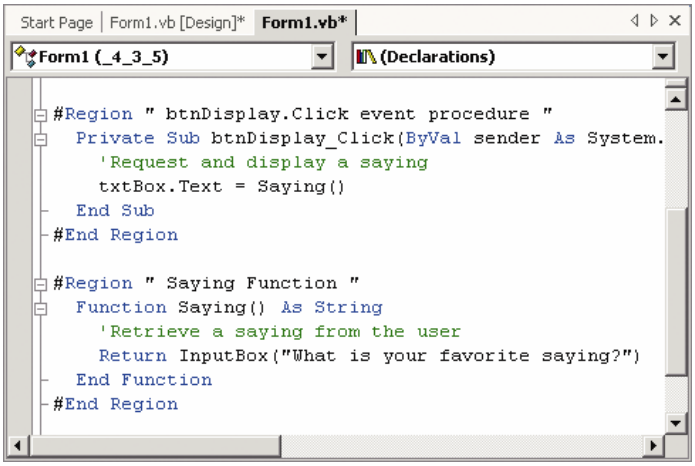
- A procedure can be collapsed behind a captioned rectangle
- This task is carried out with a **Region directive**.
- To specify a region, precede the code to be collapsed with a line of the form

#Region *"Text to be displayed in the box."*

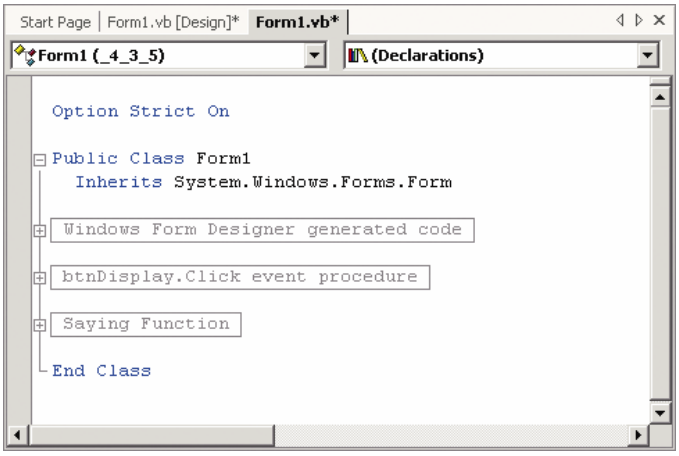
- and follow the code with the line

#End Region

Region Directives



Collapsed Regions



In-Class Exercise

- Write a function named RollDice that simulates rolling two six-sided dice and returns the sum of the roll
  - Print out several rolls to see if it is working
- Modify your function so it rolls N dice, each with M sides and returns the sum

Recursion

- Self-referential or recursive procedures: procedures that call themselves
- Direct recursion: a procedure invokes itself
- Indirect or mutual recursion: a procedure invokes a second procedure, which in turn invokes the first procedure

Mathematical Recursion

- A solution to a problem can be stated in terms of "simple" versions of itself
- Some problems can be solved using an algebraic formula that shows recursion explicitly
- For example: finding the factorial of a number *n*, denoted as *n*!, where *n* is a positive integer

  $1! = 1$

  $n! = n * (n - 1)!$ for $n > 1$

```
Public Function Factorial(ByVal num As Integer) As Integer
     If num = 1 Then
         Return 1
     Else
         Return num * (Factorial(num - 1))
     End If
End Function
```

Mathematical Recursion (continued)

- General considerations in constructing a recursive algorithm:
  - What is the first case?
  - How is the *n*th case related to the (*n* - 1) case?
- A repetitive solution is preferable if:
  - A problem solution can be expressed repetitively or recursively with equal ease
- A recursive solution is preferable if:
  - The program is easier to visualize using a recursive algorithm than a repetitive one
  - Recursion provides a much simpler solution