CSE 1530

Introduction to Computer Use II: Programming

Winter 2006 (Section M)

Topic E: Subprograms – Functions and Procedures

Wednesday, March 8 2006

Bill Kapralos

Overview (1):

- **Before We Begin**
  - Some administrative details
  - Some questions to consider
- **Function Subprogram (Cont. From Last Lecture)**
  - Function argument list
- **Procedure (or Sub) Subprogram**
  - Introduction
  - Example

Before We Begin

Administrative Details (1):

- **Lab Exercise**
  - You should be working on Ex 6-3 this week
    - Due Monday, March 13
  - Straightforward and simple but you will need Ex. 5-1 so if you haven't completed it, you should! (Ex. 5-1 is also simple)
- **Test 2 Reminder**
  - Wednesday, March 15 2006
- **Course Drop-Deadline**
  - Last day to withdraw is Friday, March 10 2006

Some Questions to Consider (1):

- What is a subprogram ?
- How many "different types" of subprograms will we consider ? And what are they ?
- Why use a subprogram ?
- What is the syntax of a function ?
- How can we treat the function name within the body of the function ?
- What must we do to the function name within the body of the function ?

Function Subprogram (cont.)

Function Argument List (1):

- **Passing Values to Subprograms**
  - There are times where we need provide certain values to a function in order for the function to complete its task
    - For example → the Format function requires two arguments: the value to be formatted and the format type – without these arguments, it cannot complete its task!
  - Can you think of a way where we can have a function that requires certain values from the caller but does not take any arguments ?

Function Argument List (2):

- **Passing Values to Subprograms (cont.)**
  - How about using global variables instead and simply have the function modify the global variables ?
    - This is bad practice → basically goes against the purpose of modularization!
    - The function now needs to access the global variables and may now be difficult to use across multiple programs
    - So given above considerations, we will make use of arguments!

Function Argument List (3):

- **Argument List in Greater Detail**
  - Any arguments we pass must have a name and be of specific data type
    - This of course does make sense since every value we use in VB is of a specific type
    - Keep in mind that the argument does not have to be the same name as the variable that is passed

Function Argument List (4):

- **Argument List in Greater Detail (cont.)**
  - An argument list is like a variable declaration with two differences
    - Syntax doesn't include the word "Dim" or "Private"
    - The argument has a value whenever the function is executed → the argument will have been assigned the value of whatever argument is specified when the function is called therefore, the function may use this value directly without having to assign the argument a value itself since it already has been supplied a value when the function is called

Function Argument List (5):

- **Argument List in Greater Detail**
  - Although the arguments do not have the key word "Private" or "Dim", each argument of the argument list is either passed by value or passed by reference
    - Each argument can also be preceded by the key word ByVal (for arguments passed by value) or ByRef (for arguments passed by reference)
  - Consider the following

```
Private Function myFunction(ByVal value1 As Integer, _
        ByRef value2 As Single)

Private Function myFunction(ByRef value1 As Integer, _
        ByRef value2 As Single)
```

Function Argument List (6):

- **Argument List in Greater Detail (cont.)**
  - Basically, when we send an argument to a function we can either send a copy of the value we specify in the argument list when calling the function or we send the actual value itself (e.g., the actual variable or memory address which stores the value)
    - When we send a copy, this is known as "call by value" → ByVal in Visual Basic
    - When we send the value itself, this is known as "call by reference" → ByRef in Visual Basic

Function Argument List (7):

- **Call By Value (ByVal)**
  - This is default for functions and if ByVal or ByRef not specified then it is assumed it is call by value
    - Do not have to explicitly provide the "ByVal" keyword when declaring functions
  - Consider the following function that takes a single argument ByVal

```
Private Function computeSum(ByVal value As Integer) As Integer
  …
Dim myValue As Integer
computeSum(myValue)
```

Function Argument List (8):

- **Call By Value (ByVal) (Cont.)**
  - Since the single argument has been declared ByVal in the function declaration, we call the function but before the function is actually called, a copy of the value represented by the variable "myValue" is created and actually passed rather than the actual variable itself
    - Within the function itself, even if the argument is altered, since we are dealing with a copy of the original variable, there will be no effect to the original variable

Function Argument List (9):

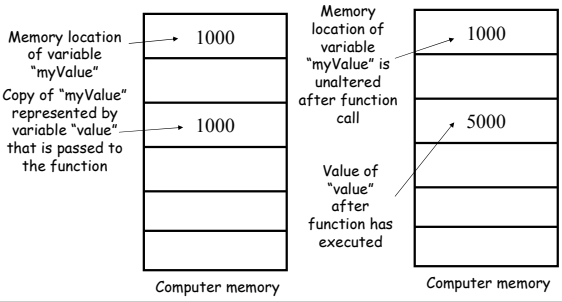
- **Call By Value (ByVal) (cont.)**
  - Consider the following code segment
    - What value will be displayed in the textbox ?

```
Private Function computeSum(ByVal value As Integer) As Integer
  value =5000
  …
End Function

Dim myValue As Integer
myValue = 1000
computeSum(myValue)
Text1.Text = CStr(myValue)
```

Function Argument List (10):

- **Call By Value (ByVal) (cont.)**
  - Graphical illustration of previous example



Function Argument List (11):

- **Call By Reference (ByRef)**
  - Consider the following function that takes a single argument ByRef

```
Private Function computeSum(ByRef value As Integer) As Integer
  …
Dim myValue As Integer
computeSum(myValue)
```

Function Argument List (12):

- **Call By Reference (ByRef) (Cont.)**
  - Since the single argument has been declared ByRef in the function declaration, when we call the function the actual value represented by the variable is passed to the function and not a copy of the value!
    - Within the function itself, since we do not have a copy of the variable but rather access the actual memory space that holds the variable's value, we can make changes that are visible even after the function returns!

Function Argument List (13):

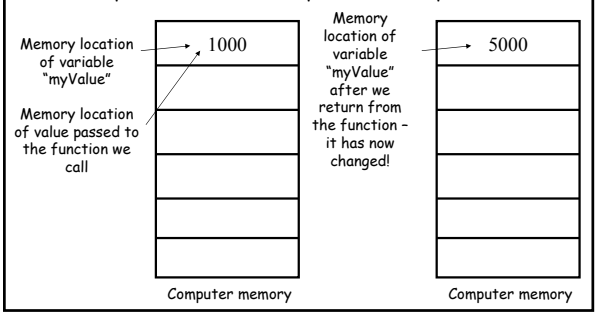
- **Call By Reference (ByRef) (cont.)**
  - Consider the following code segment
    - What value will be displayed in the textbox ?

```
Private Function computeSum(ByRef value As Integer) As Integer
  value = 5000
  ...
End Function

Dim myValue As Integer
myValue = 1000
computeSum(myValue)
Text1.Text = CStr(myValue)
```

Function Argument List (14):

- **Call By Reference (ByRef) (cont.)**
  - Graphical illustration of previous example



Function Argument List (15):

- **Specifying Multiple Arguments**
  - As you are probably already aware, a function's argument list may have either zero, one or more than one arguments
    - When there is more than one argument, for each argument, we can specify whether it is call by value or reference, the argument name and the argument type (ByVal is default)
    - Separate multiple arguments by a comma

```
Private Function computeSum(ByVal value1 As Integer, _
    ByVal value2 As Integer, ByRef value3 As Integer)
```

Function Argument List (16):

- **Some Notes**
  - Recall that in a function, we compute and return a single value after performing some operations
  - We shouldn't modify any variables of the calling program within the function or any object properties etc.
    - Therefore, values to functions will typically be "call by value" (e.g., ByVal)
    - Since ByVal is default, no need to explicitly specify it and usually, for functions we do not specify it

Procedure (or Sub) Subprograms

Introduction (1):

- **What is a Procedure Subprogram ?**
  - Recall → function computes and returns a single value
  - In contrast to a function, a procedure subprogram does not return a single result
  - Purpose is to perform some section of a larger task
    - It is said to perform some specific "sub-task" that is part of a larger task
    - Much broader role than that of a function
  - Visual Basic syntax uses the key word Sub for defining this type of subprogram

Introduction (2):

- **What is a Procedure Subprogram ? (cont.)**
  - Will generally use data available from the program that calls it and usually the data is passed via an argument list
    - Unlike a function, the values of one or more of its arguments may be changed
    - When the execution of the procedure is complete, the new values assigned to the arguments become the value of the corresponding arguments in the calling statement
    - Can also use/change the properties of objects (including control objects) if necessary

Introduction (3):

- **Defining a Procedure**
  - Very similar to a function definition except
    - No return data type specified therefore, no return value by a procedure
    - No need to assign a value to the procedure name as done with functions
  - Procedure syntax

```
Private Sub procedureName(argumentList)
  …
  procedure body
  …
End Sub
```

Introduction (4):

- **Defining a Procedure (cont.)**

```
Private Sub procedureName(argumentList)
  …
  procedure body
  …
End Sub
```

  - Private, Sub, End Sub
    - Keywords that are required
  - procedureName
    - The name you assign to the procedure

Introduction (5):

- **Defining a Procedure (cont.)**

```
Private Sub procedureName(argumentList)
  …
  Procedure body
  …
End Sub
```

  - Procedure body
    - Visual basic statements
  - Argument list
    - A list of arguments that you pass to the procedure
    - Same as functions except we must specify ByVal or ByRef for each argument

Introduction (6):

- **Using (Calling) a Procedure**
  - Similar to calling a function but some slight differences → "call" the procedure with its name and any arguments if required BUT
    - If the procedure takes no arguments, unlike calls to functions, parenthesis are not needed when making the call to it → if you do use them, it may lead to an error
    - Precede the procedure call with the keyword Call (not always necessary depending on whether the procedure has arguments) so good idea to use it!
  - The following example will illustrate this

Procedure Example (1):

- **Compute a Sum (Same as Last Lecture)**
  - Consider a function that will compute the sum of the numbers in the range 1-100 and place result in a Textbox
    - Procedure name → computeSum
    - Arguments → none
    - Return data type → none since a procedure!
  - Procedure definition

```
Private Sub computeSum()
  …
End Sub
```

Procedure Example (2):

- **Compute a Sum (cont.)**
  - Here is another (equivalent) version of the function
    - What is the difference ?

```
Private Sub computeSum()
  Dim loopIndex As Integer
  Dim sum
  sum = 0

  For loopIndex = 1 To 100
    sum = sum + loopIndex
  Next
  txtSum.Text = CStr(sum)
End Sub
```

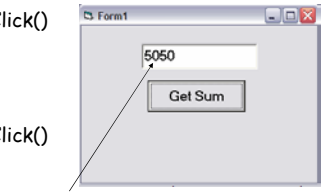
Procedure Example (3):

- **Compute a Sum (cont.)**
  - Lets use the function now
    - Call it in the button Click event handler

```
Private Sub btnSum_Click()
  Call computeSum
End Sub
```

or

```
Private Sub btnSum_Click()
  Call computeSum()
End Sub
```



Output after pressing button

Final Note (1):

- **Practice Writing Your Own Functions and Procedures**
  - Both with and without arguments
  - Practice calling them as well