PROCEDURES & ARGUMENTS PART 2

VISUAL BASIC

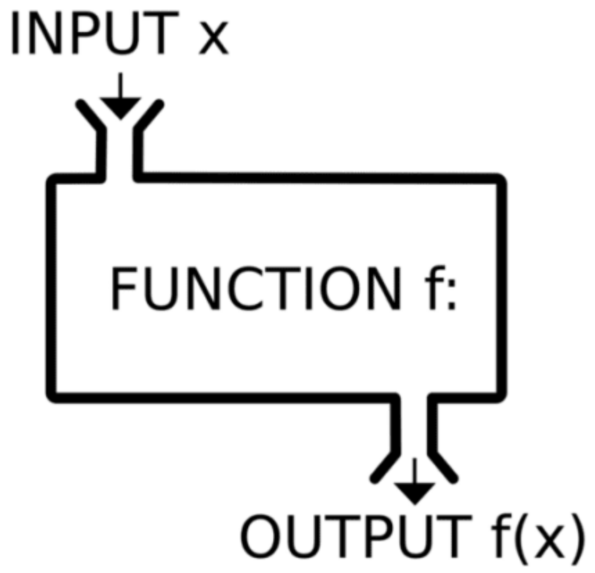
TABLE OF CONTENTS

LECTURE CONTEXT

- Most programs are quite large when one considers the total number of lines of code.
- This size links to complexity and as complexity grows, the error count grows exponentially along with the debugging time.
- . The accepted method of dealing with this is to break the program down into small sections.
- Procedures accomplish this and Functions are the second type we will look at.
- Functions allow one to pass data in and return a value and thus are often more useful than subroutines.

FUNCTIONS

- Code is broken into smaller sections for easier program design, easier debugging and more flexible programming
- They are similar to subroutines but can **return** arguments as well as receiving them as inputs
- Function names have data types associated with them (subs do not)



Declaration: Private Function *name*(inputs)as *Type*

FUNCTIONS

- As with SubRoutines, we may pass arguments in By Value or By Reference
- This (useless) example shows the typical format or ‘shape’ of a Function
- Data is returned to the calling procedure using the Return command
-

```
Private Function Reduce(ByVal intVal1 as Integer) as integer
  Dim intResult as Integer
    intResult = intVal1 - 5
  Return(intResult) ‘return what is in the brackets to calling procedure
End Function
```

INVOCATION OF FUNCTIONS

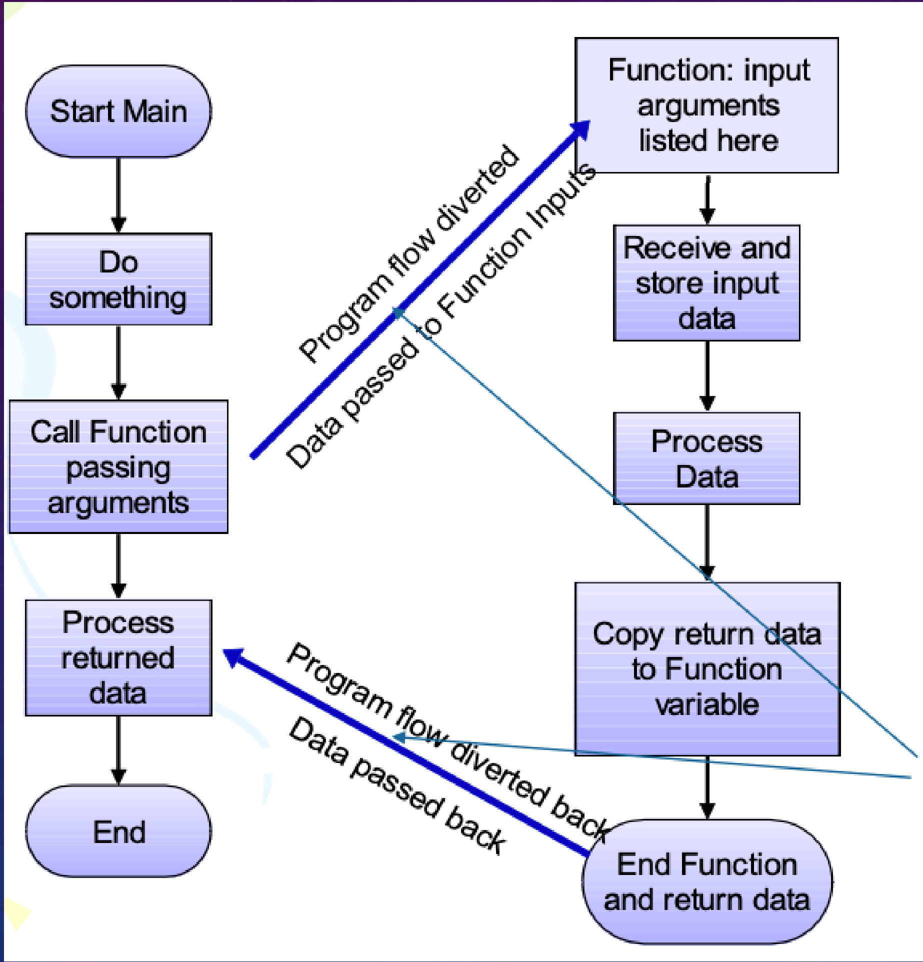
- Invocation different to subs as function returns something that we must store in a variable for further processing
- After run, intReturnedResult will contain 23-5, the same value as returned by Reduce *Function*
- Functions can return only a <u>single</u> item (may be

an array or structure)

- Data passed into function is ‘local’: variables created as procedure is run – <u>efficient resource use!</u>

intReturnedResult = Reduce(23)

HOW DOES A FUNCTION WORK ?

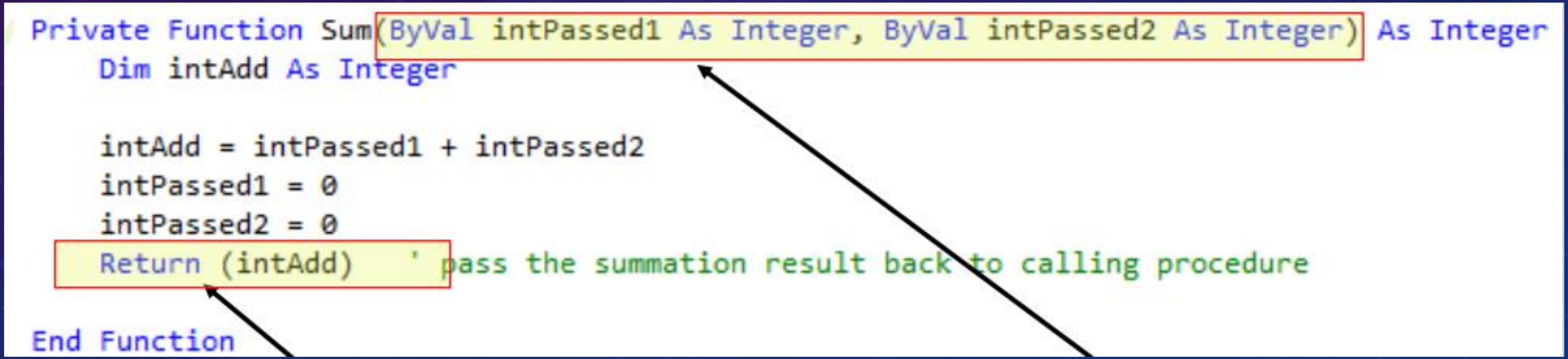

Start Main
Do something
Call Function passing arguments
Process returned data
End
Program flow diverted
Data passed to Function Inputs
Function: input arguments listed here
Receive and store input data
Process Data
Copy return data to Function variable
End Function and return data
Program flow diverted back
Data passed back

Program execution diverted to called routine – fills return variable in calling routine when the Return command is run and End Function reached

Note that these arrows are for illustration in lecture only. Do not include in your flowcharts

CREATING FUNCTIONS

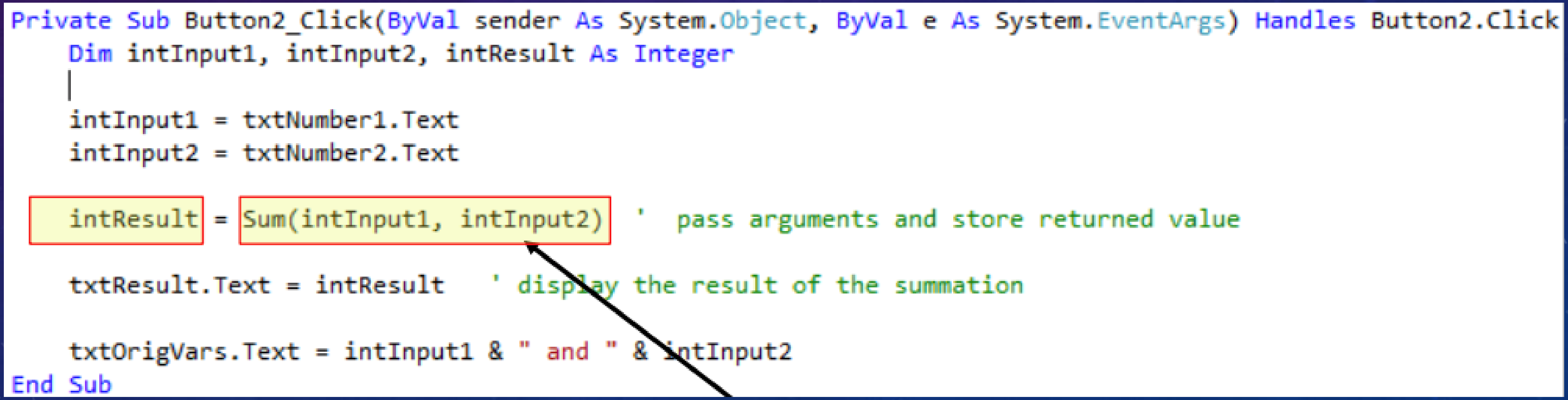
- This Function receives 2 COPIES of original variables’ contents and RETURNS one value
- Input variables are declared in the brackets as in a sub routine and receive data (ByVal) or address (ByRef) from the calling routine.
- Return value (back to the calling routine) is defined when the Return instruction is encountered



```
Private Function Sum(ByVal intPassed1 As Integer, ByVal intPassed2 As Integer) As Integer
    Dim intAdd As Integer

    intAdd = intPassed1 + intPassed2
    intPassed1 = 0
    intPassed2 = 0
    Return (intAdd)    ' pass the summation result back to calling procedure

End Function
```

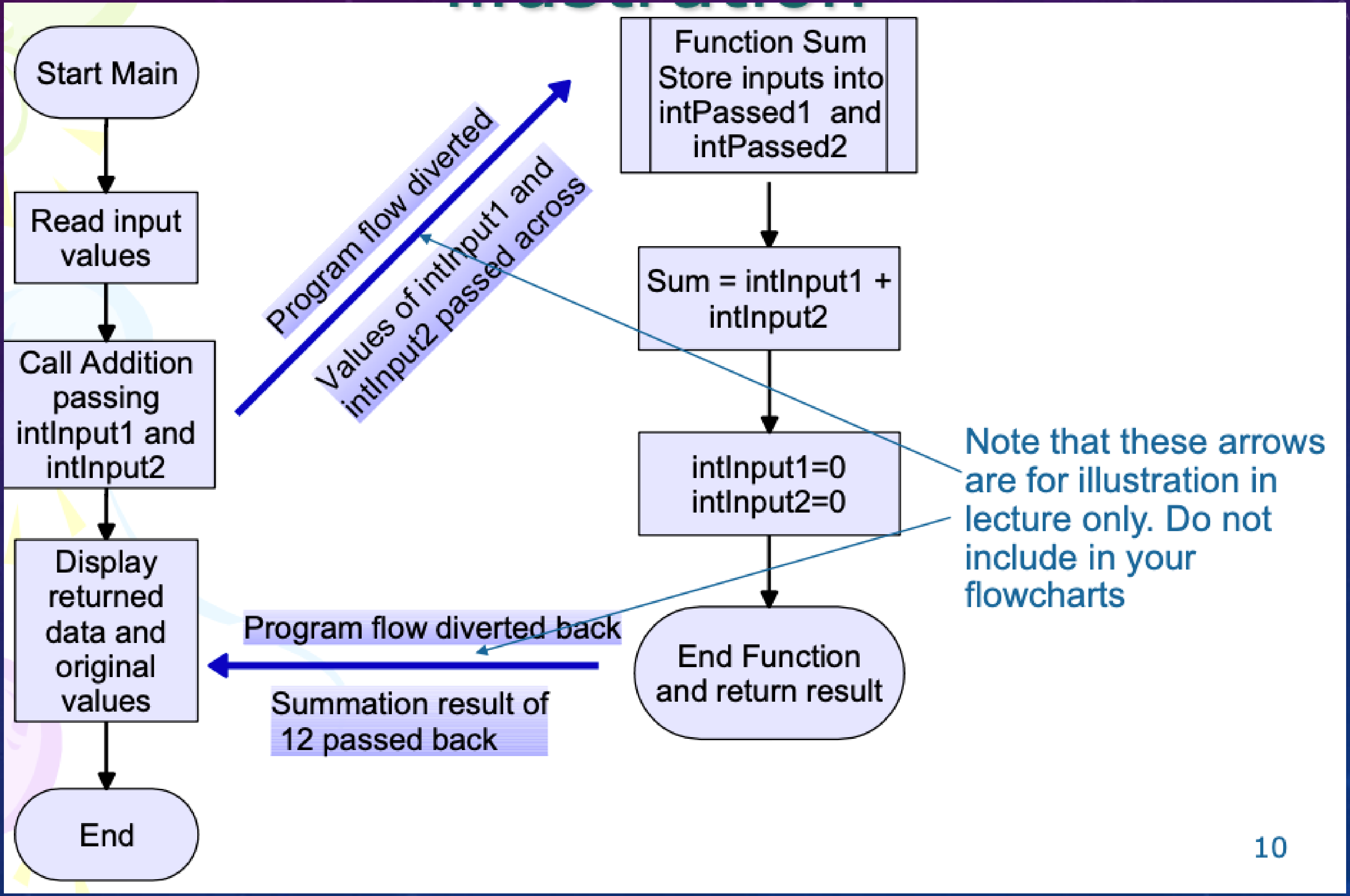
CREATING FUNCTIONS

- Passes the *intInput1 and intInput2 to the Function after reading from the textboxes*
- *Return value stored in intResult*
- After the code is run: Sum = 12 and *intVal1* = 10, *intVal2* = 2
  (original variables unchanged when using ByVal in the Function declaration)



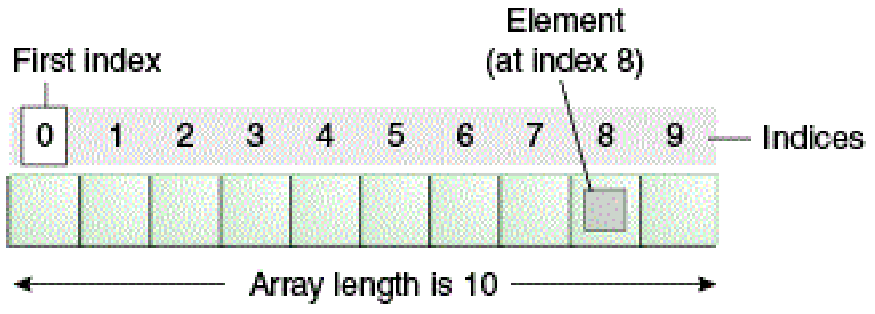
```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim intInput1, intInput2, intResult As Integer

    intInput1 = txtNumber1.Text
    intInput2 = txtNumber2.Text

    intResult = Sum(intInput1, intInput2)   '  pass arguments and store returned value

    txtResult.Text = intResult   ' display the result of the summation

    txtOrigVars.Text = intInput1 & " and " & intInput2
End Sub
```

FUNCTION OPERATION FLOW ILLUSTRATION


Start Main
Read input values
Call Addition passing intInput1 and intInput2
Display returned data and original values
End
Program flow diverted
Values of intInput1 and intInput2 passed across
Function Sum Store inputs into intPassed1 and intPassed2
Sum = intInput1 + intInput2
intInput1=0 intInput2=0
End Function and return result
Program flow diverted back
Summation result of 12 passed back
Note that these arrows are for illustration in lecture only. Do not include in your flowcharts
10

PASSING ARRAYS TO FUNCTIONS

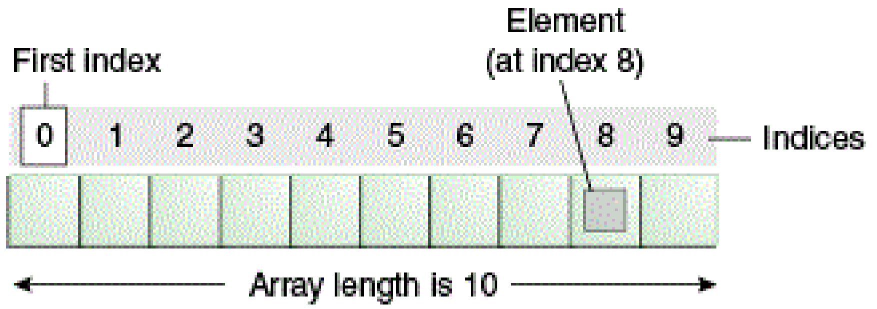
- Most often with engineering applications process groups of data, especially in instrumentation applications
- Arrays may be passed into and returned by Functions, but care must be taken with syntax
- When passing to a Function, do not use parentheses after the array name
- Input arrays must be dynamic
- The laboratory coursework requires that you pass arrays into and out of procedures



Graphical representation of Array

PASSING ARRAYS TO FUNCTIONS

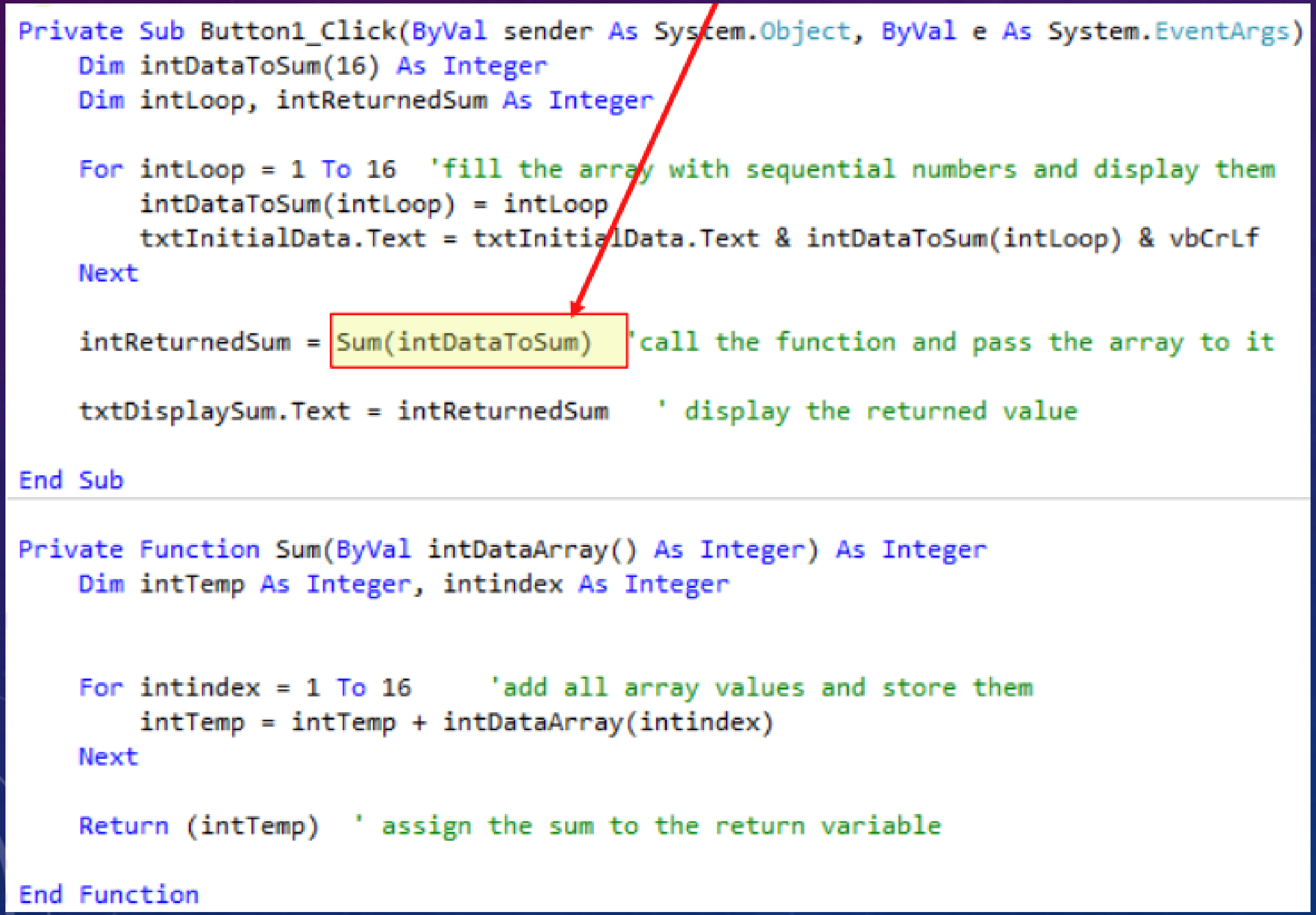
- The following example program fills an array with the values 1-16 in a For Loop construct
- It then passes this array By Value (a copy of the Array's contents) to the 'Sum' Function – note no parentheses after the array name
- Program flow passes to the Function and the array's values are summed in another For Loop
- The summation result, stored in intTemp, is returned to the calling procedure and appears in intReturnedSum (**NOTE** intTemp and IntReturnedSum must be of the same TYPE)



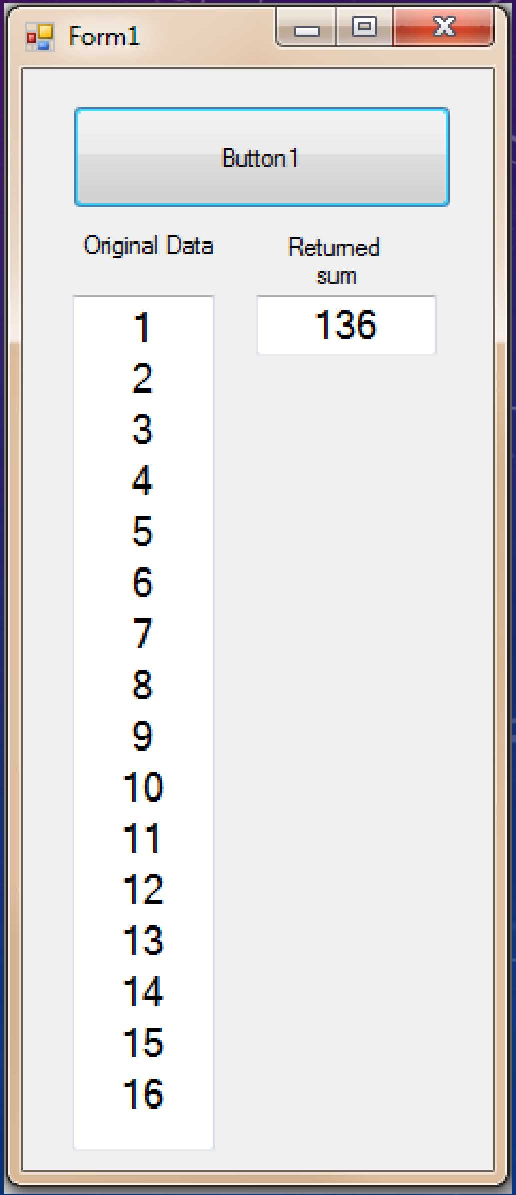
Graphical representation of Array

PASSING ARRAYS TO PROCEDURES

- Must pass an array to a procedure with empty parentheses



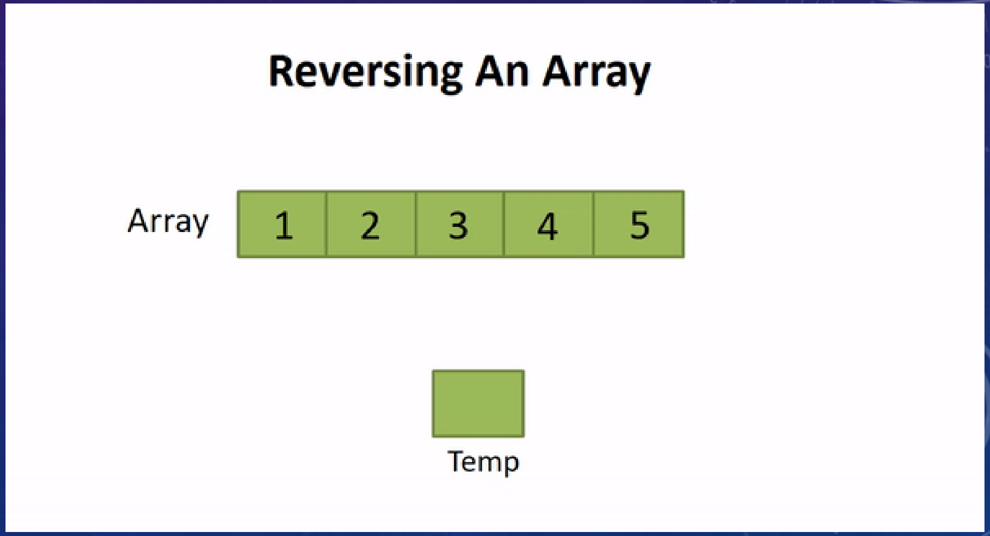
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim intDataToSum(16) As Integer
    Dim intLoop, intReturnedSum As Integer

    For intLoop = 1 To 16  'fill the array with sequential numbers and display them
        intDataToSum(intLoop) = intLoop
        txtInitialData.Text = txtInitialData.Text & intDataToSum(intLoop) & vbCrLf
    Next

    intReturnedSum = Sum(intDataToSum)  'call the function and pass the array to it

    txtDisplaySum.Text = intReturnedSum    ' display the returned value

End Sub

Private Function Sum(ByVal intDataArray() As Integer) As Integer
    Dim intTemp As Integer, intindex As Integer


    For intindex = 1 To 16      'add all array values and store them
        intTemp = intTemp + intDataArray(intindex)
    Next

    Return (intTemp)  ' assign the sum to the return variable

End Function
```



Original Data	Returned sum
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	136

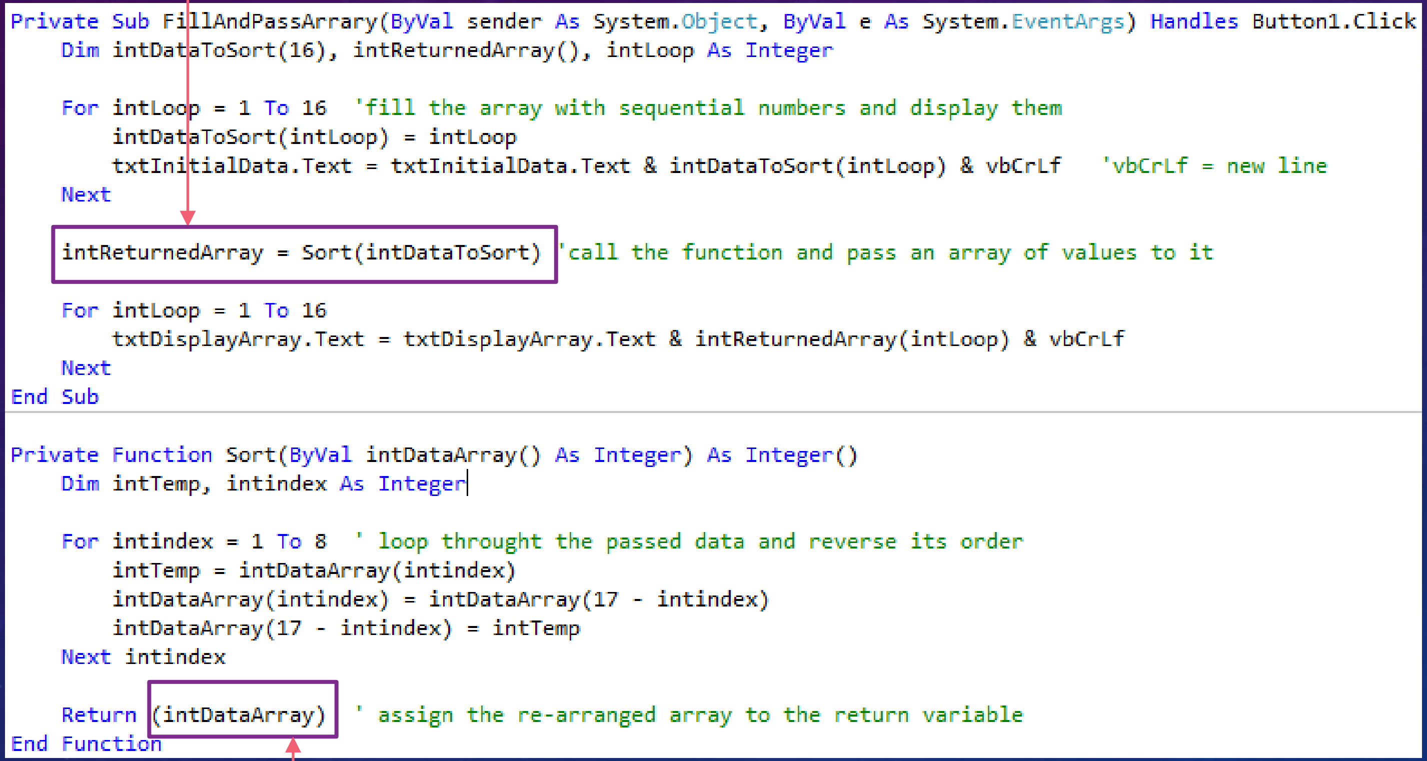
PASSING ARRAYS TO FUNCTIONS

- The next example program is identical until the Function call line which has an Array as the receptacle for returned data (no parentheses)
- The Function reverses the order of the 16 passed values in a For Loop construct by storing one in a temporary variable, overwriting it with the opposite value, then copying the temporary value to the opposite's location
- Eg. 1 is put into Temp, overwritten in the array by 16, then Temp overwrites 16 in the array
- The Function then returns the reversed array

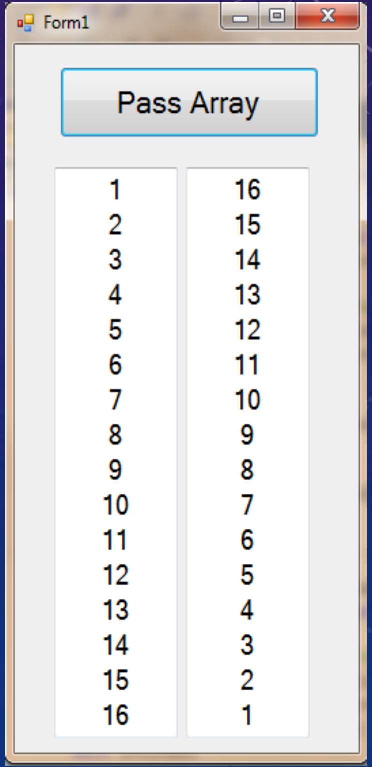


PASSING ARRAYS TO AND RETURNING FROM A FUNCTION

- Pass the array into the Function without parentheses and none on receiving variable either



```
Private Sub FillAndPassArray(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim intDataToSort(16), intReturnedArray(), intLoop As Integer

    For intLoop = 1 To 16  'fill the array with sequential numbers and display them
        intDataToSort(intLoop) = intLoop
        txtInitialData.Text = txtInitialData.Text & intDataToSort(intLoop) & vbCrLf   'vbCrLf = new line
    Next

    intReturnedArray = Sort(intDataToSort) 'call the function and pass an array of values to it

    For intLoop = 1 To 16
        txtDisplayArray.Text = txtDisplayArray.Text & intReturnedArray(intLoop) & vbCrLf
    Next
End Sub

Private Function Sort(ByVal intDataArray() As Integer) As Integer()
    Dim intTemp, intindex As Integer

    For intindex = 1 To 8  ' loop throught the passed data and reverse its order
        intTemp = intDataArray(intindex)
        intDataArray(intindex) = intDataArray(17 - intindex)
        intDataArray(17 - intindex) = intTemp
    Next intindex

    Return (intDataArray)  ' assign the re-arranged array to the return variable
End Function
```



```
 1    16
 2    15
 3    14
 4    13
 5    12
 6    11
 7    10
 8     9
 9     8
10     7
11     6
12     5
13     4
14     3
15     2
16     1
```

- Return variable has no parentheses