Outline

- Recap on Modules
- How modules communicate
- Passing information between modules
- Subroutines vs. Functions
- TEST 2 (week 11)

Module intercommunication

- Modules should be as self contained as possible
  - Focus on achieving a single task
- Called modules often are generic processes
  - How can we specify particulars of case?
- Data and control information from one module often is needed by a called module
  - How can subordinate module access data?

Module communication

- Global data/variables
  - Can be accessed in both the calling module and the called module
- Local data/variables
  - Accessible only within the module that declares it (private to module)
  - Reduces possibility of side effects (when one module alters data declared and used in another module)
  - Can be provided in the form of arguments or parameters to called modules that require their data

Passing Data via parameters

- Parameters allow information to be passed between two modules
  - One can send information to the called module
  - One can receive information back from the called module
- A Parameter is a local variable of a called (subordinate) module
  - Formal parameters are the variables defined as part of the module's declaration
  - Actual parameters are the values actually passed to the module when it is called
- The module is called by providing arguments whose types match the parameters
  - Arguments may be:
    - Variables that exist within the scope of the calling module
    - Constants
    - Result of an expression

Calling a Parameterized Module

When this line occurs:
Print_Blank_Lines ( 4 )
num_blank_lines is given the value 4
This module is called:
Print_Blank_Lines ( num_blank_lines )
DO FOR count ← 1 to num_blank_lines
Print blank line
END DO
RETURN
It will be as though this loop said ‘4’
Passing Data via Parameters

- Parameters may be passed to a module in two different ways:
  - Parameters passed by value
  - Parameters passed by reference

  - Syntax:
    ```
    ByVal/ByRef Param_name As datatype
    ```

    > ByVal keyword means parameter is passed by value
    > ByRef keyword means parameter is passed by reference

Passing ByVal

Event Procedure that calls a Sub Procedure
```vbnet
Private Sub Button1_Click(...) Handles Button1.Click
    txtOutput.Text = "Line 1 of Output"
    Call printThreeBlankLines(4)
    txtOutput.Text = txtOutput.Text & "Line 2 of Output"
    Call printThreeBlankLines(7)
    txtOutput.Text = txtOutput.Text & "Line 3 of Output"
End Sub
```

Sub Procedure:
```vbnet
Public Sub printThreeBlankLines (ByVal numberBL As Integer)
    Dim j As Integer
    For j=1 To numberBL
        txtOutput.Text = txtOutput.Text & vbCrLf
    Next j
End Sub
```

Passing byRef

- Example:
  ```vbnet
  call calcPay(10, amountEarnt)
  ```
  ```vbnet
  Sub procedure
  Private Sub calcPay ( ByVal hours As Single , ByRef earnings As Single )
      earnings = hours * 24.38
  End Sub
  ```

  - A change to `hours` will not change the argument source.
  - A change to `earnings` will change the argument source, which in the calling procedure must be a variable.

Passing by Reference

- Two way communication
  - Calling and called modules both refer to the memory location of the passed variable
  - Although, they may have different names for the memory.
  - The arguments provided in the module call must be variables
  - The values of the arguments are copied to the formal parameters in the module, but...
  - Any changes that are made to the values of the formal parameters in the called module are also made to the variables given as arguments in the calling module

Showing Parameters on Structure Charts

- Parameters are indicated as data flows on hierarchy charts

![Structure Chart](chart.png)
Parameters Programming Example

Write a program that reads two names, and outputs them in alphabetic order.

Hierarchy Chart:

```
  Process_names
    Two way communication
  Read_the_names
    Read_the_names
  Output_alphabetically
    One way communication
    Output_alphabetically
```

Solution Algorithm (Mainline):
```
Process_names
  Read_names (name1, name2)
  Output_alphabetically (name1, name2)
END
```

Parameters Programming Example (cont.)

```
Read_names (first, second)
  Prompt the user for two names
  Get first, second
END

Output_alphabetically (string1, string2)
  IF string1 < string2 THEN
    Print string1, string2
  ELSE
    Print string2, string1
  ENDIF
END
```

Values of the formal parameters are changed - data is passed by reference in this case

Values of the formal parameters are not changed - data should be passed by value in this case

Passing arrays to modules

```
MAINLINE:
  Process_Rainfall
    ReadRainfallData (rainfall_array , 12)
    CalculateAverageRainfall (rainfall_array , 12, avg_rainfall)
    Display "The average rainfall was", avg_rainfall
END
```

```
ReadRainfallAmounts ( rainfall_array , array_size)
  DO index = 0 to array_size -1
    Read rainfall_array (index)
  END DO
RETURN

CalculateAverageRainfall (rainfall_array, array_size, average)
  Set sum to zero
  DO index = 0 to array_size -1
    Set sum = sum + rainfall_array (index)
  END DO
  average = sum / array_size
RETURN
```

```
NOTE: You can use the array’s GetUpperBound method or the ( .length -1 ) to find out the upper-bound index for the array, and therefore would not need to pass the array_size into the subroutine
```

Passing arrays to procedures

```
• Brackets not needed when passing a whole array as the argument to the procedure:
  Dim intRainfallArray(5) As Integer
  ...
  ReadRainfall (intRainfallArray)
```

```
• A solitary element of an array can be given as an argument:
  Add(7, intRainfall(3))
```

```
Element intRainfall(3) is passed to sub-procedure Add
Private Sub Add(ByVal num1 As Integer, ByVal num2 As Integer)
  ...
End Sub
```
Passing Arrays Between Procedures

- Arrays can be used for parameters
- Include the brackets when declaring array parameters:
  - Arrays are special and can be passed by value or by ref

```vbnet
Private Sub ReadRainfall(ByRef intArray() As Integer)
    Dim intMonth As Integer
    ' Read each month’s rainfall and add to
    ' an accumulator (sum)
    For intMonth = 0 To intArray.Length - 1
        intArray(intMonth) = InputBox("Enter Rainfall " & intMonth)
    Next month
End Sub
```

Lecture 2: User-Defined Functions

- Special type of procedure
  - Designed to return one value
  - Intended to always return something
- Created and named by programmer
  - So really, should be called a programmer-defined function
- The sole value returned is called the return value
  - The data type of this value must be specified in the function heading
  - The last line before end of function must be a Return statement

Writing a User-Defined Function

- Functions in VB.NET have the following syntactical structure:

  ```vbnet
  Private Function Name(Parameters) As Return-Type
  __Calculation...
  Return return-value
  End Function
  ```

Example VB.NET Function

Example: Temperature Conversion Function (Celsius to Fahrenheit):

```vbnet
Private Function C_to_F(ByVal temp_c As Single) As Single
    Dim temp_f As Single
    temp_f = (9 / 5) * temp_c + 32
    Return temp_f
End Function
```

Calling/Invoking a User-Defined Function

- Functions are invoked by expressions
  - In assignment statements
    ```vbnet
    newTemp = C_to_F(200)
    ```
    - In conditions
      ```vbnet
      If waterTemp > C_to_F(50) Then
      ```
  - In places where variables or constants are normally expected
    ```vbnet
    txtOutput.Text = "Temperature is " & C_to_F(180)
    ```

The result of invoking the function is a value, which effectively replaces the function in the line of code where the call to the function appears.

When to Use a Function

- Functions are intended to calculate a single result
- Functions should not change arguments
  - So parameters should be received by value
- If a module from an algorithm calculates a single value, it should be implemented in VB.NET as a Function instead of as a Sub Procedure
TEST 1 Overview

- Test 1 covered topics weeks 1-5
- Topics include:
  - Week 1
    - Broad overview of Computer Components
    - Algorithms
    - Structure Theorem
    - Steps in writing a program
    - Desk checking an algorithm
  - Week 2
    - Objects and Classes
    - Properties and behaviours
    - VB Objects/Controls
    - VB.NET IDE
    - Built a simple splash screen
  - Week 3
    - UML Charts
    - Designing a simple interface
    - In-built functions: Val(), Format()
    - garbage Collection and Reference
    - Variables and Constants
    - Option Explicit and Implicit
    - Input/Output
  - Week 4
    - Selection
    - Statements
    - Case statements
  - Week 5
    - Repetition
    - The For Next Loop
    - The Do While Loop
    - The Repeat Until Loop

TEST 2 Overview (covers topics week 6-10)

- Week 6
  - Types of Programming Errors
    - Syntax errors
    - Runtime errors
    - Logical errors
  - Debugging Tools in VB.NET
    - Setting break points and watches
    - Stepping through code
    - Correcting code
- Week 7
  - Declaring Arrays
  - Assigning values
  - Properties and methods of the array class
  - Searching arrays
- Week 8
  - What is a file?
    - File Operations
    - StreamReader and StreamWriter
    - Exception Handling
    - List Boxes
    - String Formatting
- Week 9
  - Programming Modules (in VB.NET)
  - Module Communication (Parameters)
- Week 10
  - Parameter passing ByVal & ByRef
  - User-defined Subroutines and Functions

TEST 2 Overview

- Test will be 1 hr long
- Counts for 10% of total mark
- 4 questions
  - Q1 Programming errors and debugging [8 marks]
  - Q2 Writing a User-defined Subroutines and Functions [18 marks]
  - Q3 Arrays [10 marks]
  - Q4 File I/O, TryCatch, Listboxes [14 marks]
- Very typical of types of questions in the exam
- Wishing you all the success you deserve!

Summary/Reading

- Robertson Chapters 8 and 9
- Unit Guide, Study Guide 6
- Zak Chpt 7