Outline

- Lecture 1
  - Modules in Algorithms
- Lecture 2
  - Programming Modules in VB.NET using independent subroutines
  - Module Communication (Parameter passing)

Modular Programming

- How do we deal with large and complex programming problems?
- We need a divide and conquer strategy
- Top down design means
  - Identify the major sub-tasks
  - Divide the problem into sub-tasks - or sub-problems
  - Solve each sub-problem separately:
    > Divide into further sub-tasks
    > etc..
- Each sub-task becomes a module

Example: Entertaining Friends

Imagine you are wanting to entertain your friends for an evening. How do you go about accomplishing this?

- Decide who to invite and write and send invitations
- Prepare meals in advance
- Prepare the house (for the guests)
- Heat the food, and serve it

The above is (broadly) an algorithm of what you might do

Example: Entertaining Friends

Is the previous algorithm clear enough to understand?
- Is it detailed enough to know what to do?
  - We can (and should) provide more details on what needs to be done for each of the steps of the algorithm
  - We will need to create modules for each step of the algorithm to specify the steps involved

Example: Entertaining Friends

How do we invite guests?
For each person we want to come, we:

Write an invitation
Send the invitation
Wait for their reply
**Example: Entertaining Friends**

How do we prepare meals in advance? Do the following:
- Decide a meal to make
- Buy ingredients for meal
- Cook the meal
- Freeze/Refrigerate the meal until the day of entertaining

*Until enough food is prepared*

---

**Visualising Modular Algorithms**

- Often helpful to represent algorithm structure visually
- Hierarchy Chart shows relationship between modules
  - Use boxes to represent a module
  - Called modules are shown below their calling module
  - Lines connect modules that are linked
  - Data flows between modules can be added

---

**Reading a hierarchy chart**

The order in which the modules are begun:

- All the sub-modules must complete before their parent can complete.
  - E.G. 4 and 5 must both finish before 3 can finish.

---

**Hierarchy Chart for the Example**

---

**Attributes of a Module**

- **Attributes of a Module:**
  - Large enough to perform its task
  - Include only the operations which contribute to the task
  - Single entry and exit (runs sequentially)
  - Use a descriptive name (usually verb + noun)
  - In algorithms, mark the end of module using RETURN, rather than END

- **General guidelines:**
  - Module size is approximately one page of pseudo code (max 30 steps)
  - IF..THEN..ELSE statements should not be split across modules.

---

**Mainline Module**

- A special module that ties all modules together
- Coordinates all activity in program within a logical structure
- Readable at a high-level
Example 1 - Problem

Problem
Design an algorithm that will read two names and output them in alphabetical order.

Defining Diagram/IPO Chart

<table>
<thead>
<tr>
<th>Input</th>
<th>Processing</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>name1</td>
<td>read the names</td>
<td>name1</td>
</tr>
<tr>
<td>name2</td>
<td>Output alphabetically</td>
<td>name2</td>
</tr>
</tbody>
</table>

Example 1 – Solution

Hierarchy Chart

Process_names

Read_the_names

Output_alphabetically

Example 1 – Solution Algorithm

Mainline module
Process_names
Read_the_names
Output_alphabetically
END

Example 1 – Solution (continued)

Modules for Solution Algorithm

Read_the_names
Prompt the user for two names
Get Name1, Name2
RETURN

Output_alphabetically
IF name1 < name2 THEN
Print name1, name2
ELSE
Print name2, name1
ENDIF
RETURN

Steps involved in writing a program (revised)

1. Define the Problem – Use English
2. Outline the Solution
   1. Write a defining diagram
   2. Draw an initial hierarchy chart
   3. Design the Mainline algorithm
   4. Write algorithms for other modules
   5. Revise the hierarchy chart
3. Desk Check the Algorithms
   1. Desk check each module separately
   2. Desk check the Mainline
   Followed by the other steps already defined

Example 2 – Problem

Problem
Design an algorithm that will read data records for 8 students from a file and output the data in the form of a printed report, with a header at the top of the page.

Each line of output in the report should also show the total mark for the student in question.

The file is made up of records of the form:
Student_Name, Assign1, Assign2, Exam

The end of the file is marked by a sentinel record where the name is “XXX”.
Example 2 – Problem (cont.)

An example of a record in the file:

“Bill Jones” - Student Name
10 - Assignment 1 mark
18 - Assignment 2 mark
56 - Exam mark

An example of the sentinel record at end of file:

“XXX”

Example 2 – Problem (cont.)

The layout of the report should look like the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Assignment 1</th>
<th>Assignment 2</th>
<th>Exam</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ann Uthar</td>
<td>8</td>
<td>17</td>
<td>55</td>
<td>80</td>
</tr>
<tr>
<td>Roland Brett</td>
<td>10</td>
<td>19</td>
<td>70</td>
<td>99</td>
</tr>
<tr>
<td>Jeff Smithers</td>
<td>3</td>
<td>12</td>
<td>19</td>
<td>35</td>
</tr>
</tbody>
</table>

Exercise

- Produce a defining diagram
- Draw a hierarchy chart
- Produce solution algorithm
- Identify called modules

Lecture 2: Modules in VB.NET (using subroutines)

- Each module from algorithm will be represented as one procedure in Visual Basic .NET
- VB.NET allows a range of types of procedures:
  - Event procedures
  - User-defined (really programmer-defined)
  - Sub procedures
  - User-defined functions (NEXT WEEK)

Sub Procedures – What are they?

- One section of a program which performs a single task
  - Corresponds to one module of the algorithm
- Give it a descriptive name
  - Verb + one/two-word noun/subject
- Single entry point
- Single exit point
- Instructions flow from top to bottom (in sequence)
- Triggered by some other procedure, event or function call

Why use procedures?

- Improves readability of code
- Allows code to be re-used
- Contributes to the modularisation of a program
- Improves efficiency
  - Of program
  - Of your time
- Simplifies program maintenance and extension
Event Procedures

- Triggered by events (see week 2 lecture)
- Associated with controls (graphic objects)
- Contains a `Handles` clause
  - Describes which event and which control it responds to
- For example:
  ```vbscript
  Private Sub ProcessInfoButton_Click(...) Handles ProcessInfoButton.Click
    ... code to process the information entered in response to user clicking a button ...
  End Sub
  ```

General Syntax for Declaring a sub Procedure

```vbscript
Private Sub name_Of_Procedure ( parameters )
  ... Code goes here ...
End Sub
```

Components:
- `name_Of_Procedure` – uniquely identifies this procedure
- `parameters` – declares expected input or output data, required by procedure to do its tasks — This is optional
- `Code` – the sequence of steps for performing the module’s task

Example Problem

Write a program which places several blank lines between each line of output. There are three lines of output.

(From Study Guide 6 of the Unit Book)

Solution Using Event Procedure Only

Event Procedure containing all code:
```vbscript
Private Sub Button1_Click(...) Handles Button1.Click
  txtOutput.Text = "Line 1 of Output"
  txtOutput.Text = txtOutput.Text & vbCrLf
  txtOutput.Text = txtOutput.Text & vbCrLf
  txtOutput.Text = txtOutput.Text & vbCrLf
  txtOutput.Text = txtOutput.Text & "Line 2 of Output"
  txtOutput.Text = txtOutput.Text & vbCrLf
  txtOutput.Text = txtOutput.Text & vbCrLf
  txtOutput.Text = txtOutput.Text & vbCrLf
  txtOutput.Text = txtOutput.Text & "Line 3 of Output"
End Sub
```

Solution Using a Sub Procedure

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

Sub Procedure:
```vbscript
Public Sub printThreeBlankLines ()
  txtOutput.Text = txtOutput.Text & vbCrLf
  txtOutput.Text = txtOutput.Text & vbCrLf
  txtOutput.Text = txtOutput.Text & vbCrLf
End Sub
```

Use a Parameterized Module to specify number of blank lines

```vbscript
Module Module_Name
  [Formal] Parameter
  Procedure Print_Blank_Lines ( num_blank_lines )
    DO FOR count = 1 to num_blank_lines
      Print blank line
    END DO
  RETURN
  END Procedure
End Module Module_Name
```
Call the Parameterized Module, specifying number of blank lines

GenerateOutput

Display "First line of text"
Print_Blank_Lines ( 4 )
Display "Second line of text"
Print_Blank_Lines ( 7 )
Display "Last line of text"
END

Calling a Parameterized Module

When this line occurs:
Print_Blank_Lines ( 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'

Calling a Sub Procedure

• To make the sub procedure execute, you must Call it from some other part of the code
• The Call Statement does this
• Syntax:
  Call name_Of_Procedure ( parameters )
• The keyword Call tells computer to pause at the current place in one Sub/Event procedure and jumps to the called procedure.
  – Once the called procedure finishes, the paused procedure continues
• The keyword 'Call' is actually optional in VB.NET

Summary - 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?

Passing Data via parameters

• A Parameter is a local variable of a called (subordinate) module
  – It's existence is only defined within the called 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

Passing data using literal values as arguments

• A calling module can use literal values as arguments
  SortMoreNames
  OutputAlphabetically ( "Fred", "Sue" )
  OutputAlphabetically ( "Joe", "Anne" )
END

Subordinate module
OutputAlphabetically ( string1, string2 )
IF string1 < string2 THEN
  Print string1, string2
ELSE
  Print string2, string1
ENDIF
END

Values of the arguments are copied to the corresponding formal parameters
Passing data using variables as arguments

Demonstrate_Swap

name1 = “Kevin”
name2 = “Heidi”
Display name1

Subordinate module

Sort_alphabetically( string1, string2 )

IF string2 < string1 THEN
    temporary = string1
    string1 = string2
    string2 = temporary
ENDIF

The name Heidi is displayed

Sort_alphabetically( name1, name2 )

Display name1

END

Subordinate module

If string2 is alphabetically less than string1, their values are swapped, causing name1 and name 2 to be swapped also

The name Heidi is displayed

Passing Data via Parameters

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

• Passing by Value:
  – One way communication to the called module
  – Values of the arguments in the module call are copied to the formal parameters in the module definition

• Similar syntax as for plain Sub Procedures
  – Now need to declare parameters
    – Similar to declaring variables
    – Syntax: ByRef/ByVal Param_name As datatype
    – Example:
      Private Sub printBlankLines(ByVal numLines As Integer)
      …
      End Sub

Using Parameters in VB.NET Sub Procedures

• Similar syntax as for plain Sub Procedures
  – Now need to declare parameters
    – Similar to declaring variables
    – Syntax: ByRef/ByVal Param_name As datatype
    – Example:
      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 module must be a variable.

Showing Parameters on Structure Charts

• Example:

Using Parameters in VB.NET Sub Procedures

• ByVal keyword means parameter is passed by value
• ByRef keyword means parameter is passed by reference
• Example:
  Private Sub calculate( 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.
Parameters Programming Example

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

Hierarchy Chart:

Process_names

Read_the_names

Output_alphabetically

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

Parameters Programming Example (cont.)

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

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 not changed - data should be passed by value in this case

Parameters - Summary

- Parameters allow information to be passed between two modules
  - To send information to the called module
  - To receive information back from the called module
- Indicate parameters as data flows on hierarchy charts
- VB.NET passes parameters to sub procedures
  - by value or,
  - by reference

Summary/Reading

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