<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Solving Methodology</td>
<td>2</td>
</tr>
<tr>
<td>Top Down Design</td>
<td>2</td>
</tr>
<tr>
<td>Procedures and Functions</td>
<td>5</td>
</tr>
<tr>
<td>Sub Procedures</td>
<td>6</td>
</tr>
<tr>
<td>Sending Parameters to a Sub Procedure</td>
<td>7</td>
</tr>
<tr>
<td>Pass by Value</td>
<td>8</td>
</tr>
<tr>
<td>Pass by Reference</td>
<td>9</td>
</tr>
<tr>
<td>Function Procedures</td>
<td>10</td>
</tr>
<tr>
<td>The relationship between Formal and Actual Parameters</td>
<td>11</td>
</tr>
<tr>
<td>Reasons for using Procedures and Functions</td>
<td>12</td>
</tr>
<tr>
<td>Documenting Your Procedures and Functions</td>
<td>13</td>
</tr>
<tr>
<td>Preconditions</td>
<td>13</td>
</tr>
<tr>
<td>Postconditions</td>
<td>13</td>
</tr>
<tr>
<td>Global vs. Local Variables</td>
<td>14</td>
</tr>
<tr>
<td>Testing Your Functions</td>
<td>15</td>
</tr>
<tr>
<td>Drivers</td>
<td>15</td>
</tr>
<tr>
<td>Stubs</td>
<td>15</td>
</tr>
<tr>
<td>Top Down and Bottom Up Debugging and Testing</td>
<td>16</td>
</tr>
<tr>
<td>Top down Testing</td>
<td>16</td>
</tr>
<tr>
<td>Bottom Up Testing</td>
<td>16</td>
</tr>
<tr>
<td>Debugging Tips</td>
<td>16</td>
</tr>
</tbody>
</table>
Problem Solving Methodology

How do you go about solving a large problem?

## Divide and Conquer strategy

- Divide and Conquer strategy
- Breaks the problem into smaller and more manageable tasks.
- Encourages modular and structured programming.
- One entry point, one exit point.
- Defer the details as long as possible.

Top Down Design
Example 1: (Top Down Design of a Word Processor:)

Program
File Handling
  Read
  Write
  Directory
Editing
  Typing
  Cut
  Copy
  Paste
Formatting
  Bold
  Italic
  Underline
  Font Size and Style
  Spacing
Spell Checker
  Check Dictionary
  Suggest Words
  Add to Personal Dictionary
Printing
  Select a printer
Example 2: (Top down design of an accounting package:)

Program
  Payroll
    Check Printing
    Electronic Deposit
Account Receivable
  Invoice
Account Payable
  Check Printing
Purchasing
  Purchase Orders
Procedures and Functions

# A small module (block of code) designed to perform a specific task.

# Procedures and functions are the primary mechanism by which large programs can be subdivided to small and manageable components.

# Procedures and functions allow programmers to avoid duplicating code in different parts of the program.

Sub-programs in VB .Net can be:

# Sub Procedures (Perform an assigned task)
# Functions (Perform an assigned task, and then return a value)
There are two types of Sub Procedures in VB .Net:

1) Event Procedures  
(A collection of code associated with a GUI control. Such a procedure is invoked when an event related to the GUI occurs.)

2) Independent Procedures  
(A collection of code that can be invoked by the programmer from one or more places in the program.)

Syntax:

```vbnet
Private Sub procedureName([OptionalParameterList])  
    [Statements]  
End Sub
```

Example: (Without Parameters)

```vbnet
'Function without a Parameter  
Private Sub DisplayMessage1()  
    Console.WriteLine("Hello World")  
End Sub
```

Example: (With Parameters)

```vbnet
'Function with one Parameter  (Pass By Value)  
Private Sub DisplayMessage2(ByVal Message As String)  
    Console.WriteLine(Message)  
End Sub
```

Calling the Sub Procedure:

```vbnet
Sub Main()  
    DisplayMessage1()  
    DisplayMessage2("Hello South Bend")  
End Sub
```
Parameters can be sent to a procedure either ‘By Value’ or ‘By Reference’.

- **Pass By Value** is used if the sub-program has no reason to modify the content of the parameter.

- **Pass By Reference** is used if the sub-program is expected to modify the content of the parameter.
Example: **(Pass by Value)**

<table>
<thead>
<tr>
<th>Procedure Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Sub DisplayMessage2(ByVal Message As String)</td>
</tr>
<tr>
<td>Console.WriteLine(Message)</td>
</tr>
<tr>
<td>Or</td>
</tr>
<tr>
<td>MessageBox.Show(Message)</td>
</tr>
<tr>
<td>End Sub</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure Call</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Call</strong> DisplayMessage2(&quot;<strong>Hello World</strong>&quot;) ‘Optional use of the [call] keyword.</td>
</tr>
<tr>
<td>or</td>
</tr>
<tr>
<td>DisplayMessage2(&quot;What is your name?&quot;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
</tr>
<tr>
<td>What is your name?</td>
</tr>
</tbody>
</table>
Example: (Pass by Reference)

```vbnet
'Procedure with two parameters (passed by reference).
'This sub procedure will pass the name and age back to the
'calling program.

Private Sub GetUserInfo(ByVal strName As String, ByVal strAge As String)
    Console.Write("Enter your name: ")
    strName = Console.ReadLine()
    Console.Write("Enter your age: ")
    strAge = Console.ReadLine()
End Sub
```

```vbnet
Sub Main()
    Dim Name, Age As String
    Name = "John"
    Age = "20"

    GetUserInfo(Name, Age)        'Pass by Reference
    Console.WriteLine("Name = {0}, Age = {1}", Name, Age)

    Console.ReadLine()
End Sub
```

Output

Enter your name: Mary
Enter your age: 40

Name = Mary, Age = 40
Note:

# When **pass by value** is used, arguments can be either constants or variables.

# When **pass by reference** is used, arguments **must always be variables** (no constants or string literal is allowed).
Example: (Pass by Reference vs. Pass by Value)

Procedure Declaration

'Procedure with two parameters
'Name is passed by Reference and Age is passed by Value

Private Sub GetUserInfo(ByRef strName As String, ByVal strAge As String)
    Console.Write("Enter your name: ")
    strName = Console.ReadLine()
    Console.Write("Enter your age: ")
    strAge = Console.ReadLine()
End Sub

Procedure Call

Sub Main()
    Dim Name, Age As String
    Name = "John"
    Age = "20"
    GetUserInfo(Name, Age)
    Console.WriteLine("Name = {0}, Age = {1}", Name, Age)
    Console.ReadLine()
End Sub

Output

Enter your name: Mary
Enter your age: 40
Name = Mary, Age = 20
Functions

# Functions are similar to sub-procedures, but in addition to performing a specific task, functions also return a value.

# VB.Net has many **built in functions**:

```
X = InputBox( )
X = Pmt( )
X = CInt( )
X = CStr( )
X = IsNumeric( )
```
You can also **create your own functions**:

### Syntax:

```
Private Function functionName([OptionalParameterList]) As datatype
    [Statements]
    Return (expression)
End Function
```

### Example:

```
Private Function Cube(ByVal Number As Integer) As Integer
    Return (Number * Number * Number)
End Function
```

### Function Call

```
Sub Main()

    Dim result As Integer
    result = Cube(5)  'calling a function
    Console.WriteLine("5 raised to the power of 3 is {0}", result)

End Sub
```

### Output

```
5 raised to the power of 3 is 125
```
The relationship between Formal and Actual Parameters:

# At runtime when the function is called, the arguments are passed from the actual to formal parameters.

# The variable names in the formal parameters do not have to match the actual parameters. (however, their number, type and order have to match.)

# It is a good idea to use the same variable names (if possible) in the formal and actual parameters, since it makes your program more readable and easier to remember.
Reasons for using Procedures and Functions:

- Modularity (divide and conquer)
- Reusability (to eliminate repetition of code)
- Easier to manage
- Easier to understand

When Designing a Procedure or Function Ask Yourself:

Should I use pass by value parameters?

Should I use pass by reference parameters?
Preconditions:

# Indicates what is assumed to be true before the procedure or function is called.

Postconditions:

# Indicates the effect of the procedure or function on the rest of the program. (what should be true after the module is executed)
<table>
<thead>
<tr>
<th><strong>Global vs. Local Variables</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local Variables:</strong> Variables declared within a procedure or function are local to that sub-program.</td>
</tr>
<tr>
<td><strong>Global Variables:</strong> Variables declared above the main function are global to all other functions. Global variables are accessible by all procedures and functions except for those that declare a local variable with the same name within them.</td>
</tr>
</tbody>
</table>
Testing Your Functions

Drivers:

# A driver program is used to test the individual functions.

# The driver program will call each function sending it a set of test values to see if it produces the correct results.

Stubs:

# A simplified or empty function.

# The purpose of a stub function is to test the main program or the other functions.

# Often times a stub function simply displays a message indicating that it was called and then it returns to the calling program.
Top Down and Bottom Up Debugging and Testing

Top down Testing:

! Use of stub procedures and functions to test the overall logic of your program.

Bottom Up Testing:

! Use of drivers to test individual functions or procedures.

! Pass values to the subprogram and check the results.

Debugging Tips:

! Comment the code as you write it.

! Leave a trace of execution by printing the module name, as you enter it and later when you exit it.

! Print the values of all (input) and (input/output) parameters upon entry to the procedure.

! After returning from a subprogram, print the values returned by functions or returned through the **ByRef** parameters. Make sure they are correct.

! Use a Debugger if one is available to you.