Chapter 8
Arrays
Programming In
Visual Basic.NET
Case Structure

- Best alternative for testing a single variable or expression for multiple values
- Any decisions coded with nested If statements can also be coded using Case structure
- Case Structure is simpler, cleaner, more efficient than the nested If
Select Case General Form
(also notice indenting)

Select Case expression
   Case ConstantList
      [ code to run ]
   [ Case ConstantList
      [ code to run ] ]
   [ Case Else ]
      [ code to run ] ]
End Select

If expression=value in constant list
If expression=value in constant list
If expression<>values in any of the preceding constant lists

**Case Else is an optional clause**
Select Case - Numeric Value
Example 1

Select Case intScore
   Case Is >= 100
       lblMsg.Text = "Excellent Score"
   Case 80 to 99
       lblMsg.Text = "Very Good"
   Case 60 to 79
       lblMsg.Text = "Excellent Score"
   Case Else
       lblMsg.Text = "Improvement Needed"
End Select
Select Case - Numeric Value

Example 2

```
Select Case intListIndex
    Case 0
        HandleItemZero()
    Case 1, 2, 3
        HandleItems()
    Case Else
        HandleNoSelection()
End Select
```
Select Case - String Value Example

```vbnet
Select Case txtTeam.Text.ToUpper()
    Case "TIGERS"
        (Code for Tigers)
    Case "LEOPARDS"
        (Code for Leopards)
    Case "COUGARS", "PANTHERS"
        (Code for Cougars and Panthers)
End Select
```
Lab 1

Project Name: KeyPressCase

frmKeyPress

btnCalculate

Form1

Hours 40
Rate 25
Total $1,000.00

Calculate

txtHours

txtRate

lblTotal
Lab 1 Code

Option Strict On
Option Explicit On
...
Private Sub txtHours_KeyPress(...) Handles txtHours.KeyPress
    Select Case Asc(e.KeyChar)
        Case 48 To 57, 8
            'Do nothing
        Case Else
            e.Handled = True
    End Select
End Sub
Private Sub txtRate_KeyPress(...) Handles txtRate.KeyPress
    Select Case Asc(e.KeyChar)
        Case 48 To 57, 8
            'Do nothing
        Case Else
            e.Handled = True
    End Select
End Sub
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
    lblTotal.Text = FormatCurrency(Val(txtHours.Text) * Val(txtRate.Text))
End Sub
Sharing an Event Procedure

• If the code for multiple controls is very similar, rather than writing separate code for each, the controls can share an event procedure.

• Use the **Handles** Clause at the top of the event procedure to enable the code in a single event to be used for multiple controls.

• Example - radio buttons for selecting color have essentially the same code.
Handles Clause

• Added to the top of an event procedure to make the procedure respond to events of other controls
• Key to using the shared event is the sender argument that is passed to the event procedure
• Good technique is to declare a module level variable to store the user's selection (radio button for color in the example)
Handles Clause Example

Private Sub radBlue_CheckChanged(ByVal sender as System.Object, _
    ByVal e as System.EventArgs) _
    Handles radBlue.CheckChanged, radBlack.CheckChanged, _
           radRed.CheckChanged, radWhite.CheckChanged, _
           radYellow.CheckChanged
Lab 2

Project Name: KeyPressArray
Lab 2 Code

Option Strict On
Option Explicit On
...
Private Sub txtHours_KeyPress(...) Handles txtHours.KeyPress, txtRate.KeyPress
    Select Case Asc(e.KeyChar)
        Case 48 To 57, 8
            'Do nothing
        Case Else
            e.Handled = True
    End Select
End Sub

Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
    lblTotal.Text = FormatCurrency(Val(txtHours.Text) * __ Val(txtRate.Text))
End Sub
sender Argument

- **sender** is an object with a Name property
- Possible Problem - if you refer to **sender**.Name with Option Strict turned on, a compile error for Late Binding is generated (i.e. type cannot be determined until run time rather than at compile time)
- Solution - Before using, use CType function to convert **sender** to a specific object type instead of the generic object
CType Function

- Converts object from one type to another
- General Form
  CType (ValueToConvert, NewType)
- Example
  Dim radSelected as RadioButton
  radSelected = CType(sender, RadioButton)
  Select Case radSelected.Name
    Case "radBlue"
      . . .
Sharing Event Example

' Declare a module level variable for storing color
Dim mColorNew as Color

...  
Private Sub radBlue_CheckChanged(ByVal sender as System.Object, ByVal e as System.EventArgs) Handles radBlue.CheckChanged, radBlack.CheckChanged
    Dim radSelected as RadioButton
    radSelected = CType(sender, RadioButton)
    Select Case radSelected.Name
        Case "radBlue"
            mColorNew = Color.Blue
    End Select
End Sub

btnOK's click event sets form's BackColor to mColorNew

form's BackColor to mColorNew
Arrays

- List or series of values all referenced by the same name
- Similar to list of values for list boxes and combo boxes - without the box
- Use an array to keep a series of variable for later processing such as
  - Reordering
  - Calculating
  - Printing
Array Terms

- **Element**
  - Individual item in the array

- **Index (or subscript)**
  - Zero based number used to reference the specific elements in the array
  - Must be an integer

- **Boundaries**
  - Lower Subscript, 0 by default
  - Upper Subscript
Simple Array Example

strName Array

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Janet Baker</td>
</tr>
<tr>
<td>1</td>
<td>George Lee</td>
</tr>
<tr>
<td>2</td>
<td>Sue Li</td>
</tr>
<tr>
<td>3</td>
<td>Samuel Hoosier</td>
</tr>
<tr>
<td>4</td>
<td>Sandra Weeks</td>
</tr>
<tr>
<td>5</td>
<td>William Macy</td>
</tr>
<tr>
<td>6</td>
<td>Andy Harrison</td>
</tr>
<tr>
<td>7</td>
<td>Ken Ford</td>
</tr>
<tr>
<td>8</td>
<td>Denny Franks</td>
</tr>
<tr>
<td>9</td>
<td>Shawn James</td>
</tr>
</tbody>
</table>
Defining Arrays

• Use Dim statement to declare
• Specify the number of elements in the array as the UpperSubscript
• Each element of the array will be assigned a default value
  – Numeric has a default of 0
  – String has a default of “”
Defining Arrays - Alternate Form

• Optionally, the elements in the array may be assigned values in the Dim statement
• However, if values are assigned, you cannot declare the Upper Subscript
General Form Dim Statement for Arrays

**Dim** `ArrayName(UpperSubscript)` **as** `Datatype`

**Dim** `ArrayName( )` **as** `Datatype = {InitialValueList}`

**Dim** `ArrayName( )` **as** `Datatype`

Line continuation not shown on this slide
Dim Statement for Arrays
Examples - Default Values

**Dim strName(3) as String**

*Results in an array of 4 elements:*

strName(0), strName(1),
strName(2), strName(3)

**Dim decBalance(99) as Decimal**

*Results in an array of 100 elements:*

decBalance(0), . . . , decBalance(99)
Dim Statement for Arrays
Examples - Assigned Values

Dim strDept( ) as String = {"ACT", "MKT", "HR"}
Dim intActCode( ) as Integer = {10, 20, 30, 40}
What does VB do with the array?

- When the DIM statement for the array is processed VB sets aside room for it in memory.

- Ex: `Dim strName(3) as String`
  - VB sets aside a memory location for 4 strings

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(0)</td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td></td>
</tr>
</tbody>
</table>
Referencing Array Elements

• Use the Index(s) of the Element

<table>
<thead>
<tr>
<th>strName(row)</th>
<th>strName(0) = &quot;Sam Smith&quot;</th>
<th>strName(1) = &quot;Jill Creech&quot;</th>
<th>strName(2) = &quot;Paul Fry&quot;</th>
<th>strName(3) = &quot;Rich Wells&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0) Sam Smith</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1) Jill Creech</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2) Paul Fry</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3) Rich Wells</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ReDim Statement

- Used to redeclare the UpperSubscript/Size of an array
- Can preserve original data in the array
- General Form (basic)

\[
\text{ReDim [Preserve] ArrayName(UpperSubscript)}
\]

- Examples

\[
\text{ReDim Preserve strDept(20)}
\]
\[
\text{ReDim sdtHousewares.decSale(6)}
\]
Lab 3

Project Name: Arrays

frmArray

txtName

btnAdd

lstName

btnDisplay

© 2001 by The McGraw-Hill Companies, Inc. All rights reserved.
Lab 2 Code

Option Explicit On
Option Strict On
...
Dim strAry() As String
Private Sub btnAdd_Click(...) Handles btnAdd.Click
    Static intLoop As Integer = 0
    strAry(intLoop) = txtName.Text
    If intLoop = 5 Then
        btnDisplay.Enabled = True
        btnAdd.Enabled = False
    Else
        intLoop += 1
    End If
    txtName.Clear()
    txtName.Focus()
End Sub
Private Sub Form1_Load(...) Handles MyBase.Load
    ReDim strAry(5)
    btnDisplay.Enabled = False
End Sub
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim intCount As Integer
    lstName.Items.Clear()
    For intCount = 0 To 5
        lstName.Items.Add(strAry(intCount))
    Next
End Sub
Working with Arrays

- Use Loops to reference each element in the array
  - For / Next
  - For Each / Next
For Each / Next

- VB references EACH element of the array
- VB assigns its value to ElementName
  - The variable used for ElementName must be same datatype as the array elements or an Object datatype
- Makes one pass through the loop per element
- Use Exit For statement within loop to exit early
For Each Loop General Form

**For Each** ElementName **In** ArrayName

*Statements to execute*

**Next** [ElementName]
For Each / Next Examples

' Assumes array strName previously dimensioned
Dim strOneName As String
For Each strOneName In strName
    Debug.WriteLine(strOneName) 'Write one array element
Next strOneName

' Assumes array intTotal previously dimensioned
Dim intOneTotal As Integer
For Each intOneTotal In intTotal
    intOneTotal = 0 ' reinitialize the array
Next intOneTotal
Structures

• Combine multiple fields of data into a single unit

• Declaration (by default a Structure is Public)
  – Cannot be declared inside a procedure
  – Generally declared in General Declarations

• Define using Structure, End Structure

• Once created, declare variable of the Structure as if it were another datatype; make up a meaningful prefix
Structure/End Structure
General Form

[Public|Private] Structure NameOfStructure
Dim FirstField As Datatype
Dim SecondField As Datatype
...
End Structure
Structure Example 1

Public **Structure Employee**

Dim `intEmpID` As Integer
Dim `strLName` As String
Dim `strFName` As String
Dim `datHireDate` As Date

End Structure

' Declaring a variable based on the Structure
Dim `empOffice` As `Employee`
Public **Structure** *Product*

Dim *strDesc* As String
Dim *strProdID* As String
Dim *intQuan* As Integer
Dim *decPrice* As Decimal

**End Structure**

' Declaring a variable array based on the Structure
Dim *prdInven*(100) As *Product*
Accessing the Elements in a Structure Variable

- Each field of data in Structure is an Element
- To access specify `Variable.Element`
- Examples
  
<table>
<thead>
<tr>
<th>Structure</th>
<th>Array Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>empOffice.intEmpID</td>
<td>prdInven(intIndex).strDesc</td>
</tr>
<tr>
<td>empOffice.strFName</td>
<td>prdInven(intIndex).strID</td>
</tr>
<tr>
<td>empOffice.strLName</td>
<td>prdInven(intIndex).intQuan</td>
</tr>
<tr>
<td>empOffice.datHireDate</td>
<td>prdInven(intIndex).decPrice</td>
</tr>
</tbody>
</table>
Including An Array In A Structure

- Arrays can be included as elements within a Structure
- VB does not, however, allow you to declare the number of elements in the array within the Structure declaration
- Use the ReDim statement inside a procedure to define the size of the array
Lab 4

Project Name: Structures

frmStructure

txtName

txtAddress

txtCity

btnAdd

Name

Address

City

Add

Display

lstName

btnDisplay
Option Explicit On
Option Strict On

Private Structure typEmp
    Dim Name As String
    Dim Address As String
    Dim City As String
End Structure

Dim typEmpAry() As typEmp

Private Sub btnAdd_Click(...) Handles btnAdd.Click
    Static intLoop As Integer = 0
    typEmpAry(intLoop).Name = txtName.Text
    typEmpAry(intLoop).Address = txtAddress.Text
    typEmpAry(intLoop).City = txtCity.Text
    If intLoop = 5 Then
        btnDisplay.Enabled = True
        btnAdd.Enabled = False
    Else
        intLoop += 1
    End If
    txtName.Clear()
    txtAddress.Clear()
    txtCity.Clear()
End Sub
Private Sub Form1_Load(...) Handles MyBase.Load
    ReDim typEmpAry(5)
    btnDisplay.Enabled = False
End Sub

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim intCount As Integer
    lstName.Items.Clear()
    For intCount = 0 To 5
    Next
End Sub
Using Array Elements for Accumulators (p 344 -346)

mintTotal array

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

intGroupNum = CInt(txtGroup.Text) - 1
Debugging Array Programs

- View the array elements in Break Time in the Autos Window or the Locals Window
Table Lookup (p346 - 350)

- Problem: Often values used to identify a series of elements are not
  - Numerically sequential
  - Numerically separated by a constant value
  - Numeric at all, may be strings

- Solution: Create another array to hold the identifying values and use a table lookup process to find the correct element in the array
Table Lookup (cont.)

mgiGroup array

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(0)</td>
<td>101</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td>103</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td>110</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td>115</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td>121</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5)</td>
<td>123</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6)</td>
<td>130</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(7)</td>
<td>145</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

intIndex=2
Using List Boxes With Arrays

- Use List Boxes or Combo Boxes rather than using text boxes for the user to enter data used for looking up information in the array.
- Use the list's SelectedIndex property as the subscript of the corresponding array.
List Boxes With Arrays (cont.)

mgiGroup array

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>101 0</td>
</tr>
<tr>
<td>1</td>
<td>103 0</td>
</tr>
<tr>
<td>2</td>
<td>110 10</td>
</tr>
<tr>
<td>3</td>
<td>115 0</td>
</tr>
<tr>
<td>4</td>
<td>121 0</td>
</tr>
<tr>
<td>5</td>
<td>123 0</td>
</tr>
<tr>
<td>6</td>
<td>130 0</td>
</tr>
<tr>
<td>7</td>
<td>145 0</td>
</tr>
</tbody>
</table>

SelectedIndex = 2 of List Box
Multidimensional Arrays

- Arrays can have more than one dimension
- Like a table of values
- You must specify the boundaries for each dimension using subscripts
- Example: Two Dimensional Array

Dim intScoreBoard (1, 8) as Integer
Referencing Elements in Multidimensional Array

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
0 & (0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (0,5) & (0,6) & (0,7) & (0,8) \\
1 & (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (1,5) & (1,6) & (1,7) & (1,8) \\
\end{array}
\]

\[
\begin{align*}
\text{intScoreBoard}(0,0) & \quad \text{intScoreBoard}(1,0) \\
\text{intScoreBoard}(0,1) & \quad \text{intScoreBoard}(1,1) \\
\text{intScoreBoard}(0,2) & \quad \text{intScoreBoard}(1,2) \\
\text{intScoreBoard}(0,3) & \quad \text{intScoreBoard}(1,3) \\
\text{intScoreBoard}(0,4) & \quad \text{intScoreBoard}(1,4) \\
\text{intScoreBoard}(0,5) & \quad \text{intScoreBoard}(1,5) \\
\end{align*}
\]
General Form Dim Statement for Two-Dimensional Arrays

```
Dim ArrayName(Rows, Cols) as Datatype

Dim ArrayName( , ) as Datatype = {ListOfValues}
```

Line continuation not shown on this slide
Dim Statement for Two Dimensional-Arrays Examples

Dim strName(2, 3) as String

Results in an array of 12 elements:
3 rows: 0, 1, 2, 3
4 columns: 0, 1, 2, 3, 4

Dim strName( , ) as String = {
    "Jim", "Mary", "Sam", "Sean",
    "Tom", "Sue", "Fred", "Paul",
    "Tim", "Al", "Bob", "Pete",
    "Joy", "Wes", "Kim", "Beth"
}

Results in same array as above with default values

Line continuation not shown on this slide
Working With Two-Dimensional Arrays

- Initializing/Reinitializing
  - Use Nested For/Next Loops
  - Use For Each/Next Loop

- Printing
  - Use For Each/Next Loop

- Summing
  - Include a total field for each row and each column
  - Use For/Next Loop to calculate sums
Initializing For/Next Example

Dim intRow As Integer
Dim intColumn As Integer
For intRow = 0 to 2
    For intColumn = 0 to 3
        strName(intRow, intColumn) = " "
    Next intColumn
Next intRow
Initializing For Each/Next Example

Dim strElement As String
For Each strElement In strName
    strElement = " "
Next strElement
Printing Example

'Print one name per line
For Each strElement In strName
    'Set up a line
    e.Graphics.DrawString(strElement, fntPrintFont, _
        Brushes.Black, sngPrintX, sngPrintY)
    'Increment the Y position for the next line
    sngPrintY += sngLineHeight
Next strElement
Summing Example

decAmount(3, 5)

decRowTotal(3)

decColTotal(5)
'Crossfoot Total a 2D table
Dim decAmount(3,5) As Decimal
Dim decRowTotal(3) As Decimal
Dim decColTotal(5) As Decimal
Dim intRowIndex As Integer
Dim intColIndex As Integer

For intRowIndex = 0 to 3
    For intColIndex = 0 to 5
        decRowTotal(intRowIndex) += decAmount(intRowIndex, intColIndex)
        decColTotal(intColIndex) += decAmount(intRowIndex, intColIndex)
    Next intColIndex
Next intRowIndex
Lookup Two-Dimensional Tables

• Use same techniques as for single dimension arrays
  – Direct Reference (if meaningful row and column subscripts are available)
  – Table Lookup

• Many 2D tables used for lookup will require additional one-dimensional arrays or list boxes to aid in the lookup process
Lookup Example Using List Box (p 356)

Weight Subscript of Array uses lstWeight.SelectedIndex

Zone Subscript of Array uses lstZone.SelectedIndex