Selection: Boolean Expressions

- Boolean operators:
  - Return a Boolean value
  - Types:
    1. Relational operators
       * Operators:
         (a) < (less than)
         (b) > (greater than)
         (c) = (equal to)
         (d) <= (less than or equal to)
         (e) >= (greater than or equal to)
         (f) <> (not equal to)
       * Arguments must be type-compatible
       * Note that assignment operator and equivalence test use same operator
       * When using inequalities:
         (a) Numeric types as you would expect
         (b) Character comparison based on position in \textit{collating sequence} (ASCII)
         (c) String comparison uses dictionary-style
         (d) Date comparisons based on calendar sequence
         (e) \textit{False} < \textit{True}
    2. Logical operators
       * Operators:
         (a) And
         (b) Or
         (c) Not
         (d) Xor (\textit{exclusive or})
       * Arguments must be Boolean
       * Values represented by \textit{truth tables}
         * Graphical way of expressing value of Boolean expression

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A And B</th>
<th>A Or B</th>
<th>A Xor B</th>
<th>Not A</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
Selection: Boolean Expressions (2)

– Precedence chart:

<table>
<thead>
<tr>
<th>operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>highest (</td>
</tr>
<tr>
<td>∧</td>
</tr>
<tr>
<td>unary +, -</td>
</tr>
<tr>
<td>*, /</td>
</tr>
<tr>
<td>+, -</td>
</tr>
<tr>
<td>relational operators</td>
</tr>
<tr>
<td>Not</td>
</tr>
<tr>
<td>And, AndAlso</td>
</tr>
<tr>
<td>Or, OrElse</td>
</tr>
<tr>
<td>lowest Xor</td>
</tr>
</tbody>
</table>

• Boolean functions:

1. *String* functions:
   (a) .EndsWith(*str*): ”hello”.EndsWith(”lo”)
   (b) .StartsWith(*str*): ”hello”.StartsWith(”hell”)
   (c) .Equals(*str*): ”hello”.Equals(”hello”)
   (d) .CompareTo(*str*): ”hello”.CompareTo(”hello”)
      – Returns 0 if caller equals *str*
      – Returns -1 if caller < *str*
      – Returns +1 if caller > *str*
   (e) IsNumeric(*str*): IsNumeric(”$5,000,123”)

2. *Char* functions
   (a) .IsDigit(*c*)
   (b) .IsLetter(*c*)
   (c) .IsLetterOrDigit(*c*)
   (d) .IsLower(*c*)
   (e) .IsUpper(*c*)
   (f) .IsPunctuation(*c*)
   (g) .IsSymbol(*c*) (e.g., operators: +, =, )
   (h) .IsWhiteSpace(*c*)

• Note: .Equals and .CompareTo available for all data types
• Short circuit evaluation
  – Consider the Boolean expression $a \text{ And } b$
    * If $a$ is $False$, there is no need to evaluate $b$
  – VB provides two additional Boolean operators that perform short circuit evaluation
    * They only evaluate as much as is needed to determine the truth value of the expression
  – $AndAlso$ and $OrElse$ have the same semantics as $And$ and $Or$, respectively, but use short circuit evaluation
Selection: If-Then-Else

• Basic selection statement

• Syntax (limited):
  – If Boolean-expression Then
    statements
  Else
    statements
  End If

• Semantics:
  – If Boolean-expression is True, execute statements after Then
  – If Boolean-expression is False, execute statements after Else

• Note that the Else component is optional
Selection: If-Then-Else Examples

- **Simple If-Then:**
  
  \[
  \text{pay} = \text{hours} \times \text{rate} \\
  \text{If } \text{hours} > 40 \text{ Then} \\
  \quad \text{pay} = \text{pay} + (\text{hours} - 40) \times \text{rate} \times 0.5 \\
  \text{End If}
  \]

- **Simple If-Then-Else example:**
  
  \[
  \text{If } \text{hours} \leq 40 \text{ Then} \\
  \quad \text{pay} = \text{hours} \times \text{rate} \\
  \text{Else} \\
  \quad \text{pay} = 40 \times \text{rate} + ((\text{hours} - 40) \times 1.5 \times \text{rate}) \\
  \text{End If}
  \]

- **Nested If-Then-Else example:**
  
  \[
  \text{If } \text{temp} < 100 \text{ Then} \\
  \quad \text{txtWarn.Text} = "WARNING: Temperature below normal" \\
  \text{Else} \\
  \quad \text{If } \text{temp} > 200 \text{ Then} \\
  \quad \quad \text{txtWarn.Text} = "WARNING: Elevated temperature" \\
  \text{Else} \\
  \quad \quad \text{txtWarn.Text} = "Normal operating temperature" \\
  \text{End If} \\
  \text{End If}
  \]
Selection: If-Then-ElseIf

- The complete syntax for If is

```
If Boolean-expression$_1$ Then
    statements$_1$
ElseIf Boolean-expression$_2$ Then
    statements$_2$
...
Else
    statements$_n$
End If
```

- Semantics:
  - If Boolean-expression$_1$ is True, execute statements$_1$
  - If Boolean-expression$_1$ is False, check Boolean-expression$_2$ ...

- This format much more convenient than using separate nested ifs
• *If-Then-ElseIf* example:

```vbnet
If temp < 100 Then
    txtWarn.Text = "WARNING: Temperature below normal"
ElseIf temp > 200 Then
    txtWarn.Text = "WARNING: Elevated temperature"
Else
    txtWarn.Text = "Normal operating temperature"
End If
```
Selection: If-Then-Else Style

• Improper use of nested *If-Then-Else*:
  How many exemptions does someone over 65, married to someone over 65 and filing jointly receive?

    Dim married As Boolean
    Dim age, spouseAge, dependents, exemptions As Integer
    Dim filingStatus As Char
    ...

    If age > 65 Then
      exemptions = exemptions + 1
    ElseIf spouseAge > 65 Then
      exemptions = exemptions + 1
    ElseIf (dependents > 0) Then
      exemptions = exemptions + dependents
    ElseIf married And (filingStatus = "J") Then
      exemptions = exemptions + 1
    End If

• Proper version of nested *If-Then-Else*:

    ...
    If age > 65 Then
      exemptions = exemptions + 1
    End If
    If spouseAge > 65 Then
      exemptions = exemptions + 1
    End If
    If (dependents > 0) Then
      exemptions = exemptions + dependents
    End If
    If married And (filingStatus = "J") Then
      exemptions = exemptions + 1
    End If
Selection: If-Then-Else Style (2)

• Improper use of sequential ifs:
  What letter grade does someone with a 95 get?

    If avg >= 90 Then
    grade = "A"
    End If
    If avg >= 80 Then
    grade = "B"
    End If
    If avg >= 70 Then
    grade = "C"
    End If
    If avg >= 60 Then
    grade = "D"
    End If
    If avg < 60 Then
    grade = "F"
    End If

• Better use of sequential ifs

    If avg >= 90 Then
    grade = "A"
    End If
    If (avg >= 80) And (avg < 90) Then
    grade = "B"
    End If
    If (avg >= 70) And (avg < 80) Then
    grade = "C"
    End If
    If (avg >= 60) And (avg < 70) Then
    grade = "D"
    End If
    If avg < 60 Then
    grade = "F"
    End If
Selection: If-Then-Else Style (3)

- Best use of nested ifs

```plaintext
If avg >= 90 Then
    grade = "A"
ElseIf avg >= 80 Then
    grade = "B"
ElseIf avg >= 70 Then
    grade = "C"
ElseIf avg >= 60 Then
    grade = "D"
Else
    grade = "F"
End If
```
Selection: Select Case

- *Select* statement is an alternative to nested *If-ElseIf*

- Syntax:

  ```
  Select Case expression
    Case value-list
      statements
    Case value-list
      statements
    ...
    Case Else
      statements
  End Select
  ```

- *value-list* is a comma-separated list of *values* (if more than one value)

- A *value* may have any of the following forms
  Let a *value-returning-expression* be either a literal, variable, or expression

  1. *value-returning-expression*
  2. Is relop *value-returning-expression*
  3. *value-returning-expression*_1 To *value-returning-expression*_2 where
     *value-returning-expression*_1 < *value-returning-expression*_2

- Semantics:

  - *expression* is evaluated
  - The *Case* clauses are evaluated one by one until the value of *expression*
    meets the condition of *value-list*
  - When a *Case* is True, the statements following it are executed
  - If no *Case* is True, either the statements following the *Else* are executed,
    or execution continues to the statement following the *End Select*
Selection: Select Examples

• Simple example:

Select Case year Mod 4
  Case 0
    MessageBox.Show("This is an election year")
  Case 1
    MessageBox.Show("Last year was an election year")
  Case 2
    MessageBox.Show("Two years ago was an election year")
  Case 3
    MessageBox.Show("Next year is an election year")
End Select

• Another example:

Select Case avg
  Case Is >= 90
    grade = "A"
  Case Is >= 80
    grade = "B"
  Case Is >= 70
    grade = "C"
  Case Is >= 60
    grade = "D"
  Case Else
    grade = "F"
End Select

• And another:

Select Case temp
  Case 100 To 200
    MessageBox.Show("Normal operating temperature")
  Case Is < 100
    MessageBox.Show("WARNING: Temperature below normal")
  Case Else
    MessageBox.Show("WARNING: Elevated temperature")
End Select
Selection: Input Based on User Selection

- The following examines several controls that allow the user to select from a set of alternatives provided by the program

1. List Box
   - Allows programmer to provide a list of choices for the user to select from (like a menu)
   - To add items to a List Box, select String Collection Editor from the control’s task button
     - To add them during program execution, assign string values using ListBox.Items.Add(str)
     - Selection is made by clicking on an item
       * When clicked, the string value is assigned to the List Box Text property
     - The List Box items are numbered by VB (zero-based)
Selection: Input Based on User Selection (2)

- Properties:
  (a) `.SelectedItem` returns a reference to the item that was selected
      * If value is `Nothing`, no item was selected
      * If `.SelectedItem` is assigned `Nothing`, any selection is cleared (deselected)
  (b) `.SelectedIndex` returns the index of the selected item
      * If value is `-1`, no item was selected
  (c) `.Sorted`
      * If value is `True`, items are displayed in alphabetic order
  (d) `.SelectionMode` determines how many items can be selected at one time
      * If value is `None`, nothing can be selected
      * If value is `One`, only one item can be selected at any one time
- Methods:
  (a) `.ClearSelected()` clears all selections
- Events:
  (a) `SelectedIndexChanged` is triggered when an item is selected
- List boxes tend to be used when there are many options for the user to choose from

2. Group Box

- A `Group Box` is a container that holds other objects (controls) as a group
- These objects are said to be `attached` to the group box
- To add controls to a group box, simply drag them into it
- Events are not usually associated with a group box
Selection: Input Based on User Selection (3)

3. Radio Button
   - Displayed as a circle and label
   - Usually displayed in a Group Box
   - Radio buttons in a group box are mutually exclusive - only one can be set at any one time
   - Cannot be toggled by reclicking
   - Can associate an access key
   - Properties:
     (a) .Text: label that is displayed
     (b) .Checked: Boolean that indicated whther the button is set
     * Can provide the user a default by initially setting this to True
   - Events:
     (a) CheckChanged is triggered when the button is clicked
   - Generally used for short lists (seven or fewer options)

4. Check Box
   - Displayed as a square and label
   - Usually displayed in a Group Box
   - Can select a many as you like at any one time
   - Can be toggled by reclicking
   - Can associate an access key
   - Properties:
     (a) .Text: label that is displayed
     (b) .Checked: Boolean that indicated whther the check box is set
     * Can provide the user a default by initially setting this to True
   - Events:
     (a) CheckChanged is triggered when the button is clicked
Selection: Input Based on User Selection (4)

• General note:
  – Each of the above can have events associated it
    ∗ In such a situation, the event will be triggered as soon as the user interacts with that control
  – Frequently want to allow the user to make multiple selections using a combination of controls
    ∗ In this case, would not want to associate events with the individual controls
    ∗ Rather, provide a button for the user to click once the user has finished making selections

• Processing of the above controls is achieved using selection, as the program must check each control to determine which list box selection, radio button, and check box selections were set by the user