Additional Controls and Making Decisions (Chapter 4)
CS109

In this lecture we will briefly examine a few new controls and how to make if-then-else statements. The textbook covers most of these in chapter 4.

**Combo Box**

The Combo Box control is like a textbox with a pull-down menu of choices. We can access the user’s selection with:

```
comboBox.Text
```

We can add to the items just like with a listbox:

```
comboBox.Items.Add(newItem)
```

(Short demo in class)

**Group Box Control**

The group box is used to group related sets of controls for visual effect. To use it, drag the group box onto the form. Then drag any new controls into the group box. The new controls will now be “part” of the group box.

The group box can be used to create different sets of radio buttons (upcoming).

(Short demo in class)

**Check Box Control**

The checkbox is a small box that can be checked or unchecked by the user. To see if something is checked or not you can inspect the “Checked” Property:

```
checkbox.Checked - True if checked, False if not
```

(Short demo in class)

**Radio Button Control**

The radio button operates like an old car radio. When one button is pushed, any other buttons “pop out”. For all radio buttons that are on a form, only one can be active at a time. If you would like to have multiple subgroups of radio buttons then they should be added to a GroupBox.
To see the value of a radio button, you can inspect the .Checked property just as with a checkbox.

(Short demo in class)

**Main Menu Control**

This control allows you to add a menu to the application. To use it, drag a Main Menu control to your form. Then double-click it in the form area. A menu designer will appear at the top of your form saying “Type Here”.

You can now type the name of the top-level of your menu. Click and type to fill in sub-areas. To attach code to the sub-areas, double-click on the menu item. The VB Code window will appear with an event for you to fill in code.

Try adding a menu for F)ile, O)pen, and C)lose.

**Boolean Expressions and Conditions**

The physical order of a program is the order in which the statements are *listed*. The logical order of a program is the order in which the statements are *executed*. With conditional structures and control structures that we will examine soon, it is possible to change the order in which statements are executed.

**Boolean Data Type**

To ask a question in a program, you make a statement. If your statement is true, the answer to the question is yes. If your statement is not true, the answer to the question is no. You make these statements in the form of *Boolean expressions*. If you recall from the previous lecture, a Boolean expression asserts (states) that something is true. The assertion is evaluated and if it is true, the Boolean expression is true. If the assertion is not true, the Boolean expression is false.

In VB.NET, the data type **Boolean** is used to represent Boolean data. Each **boolean** constant or variable can contain one of two values: **True** or **False**.
Relational Operators

A Boolean expression can be a simple Boolean variable or constant or a more complex expression involving one or more of the relational operators. Relational operators take two operands and test for a relationship between them. The following table shows the relational operators and the VB.NET symbols that stand for them.

<table>
<thead>
<tr>
<th>VB.NET Symbol</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to (since there is no ≠ symbol on the keyboard)</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

For example, the Boolean expression

\[
\text{number1} < \text{number2}
\]

is evaluated to True if the value stored in number1 is less than the value stored in number2, and evaluated to False otherwise.

Examples:

```vbnet
Dim b as Boolean
b = 3 < 1
Console.WriteLine(b)  ' Outputs “false”

b = 3 > 1
Console.WriteLine(b)  ' Outputs “true”
```

When a relational operator is applied between variables of type String, the assertion is in terms of where the two operands fall in the collating sequence of a particular character set. For example,

```
character1 < character2
```

is evaluated to true if the character stored in character1 comes before the character stored in character2 in the collating sequence of the machine on which the expression is being evaluated. Although the collating sequence varies among machines, you can think of it as being in alphabetic order. That is, A always comes before B and a always before
We must be careful when applying the relational operators to floating point operands, such as doubles, particularly equal (=) and not equal (<>). Integer values can be represented exactly; floating point values with fractional parts often are not exact in the low-order decimal places. Therefore, you should compare floating point values for near equality. For now, do not compare floating point numbers for equality. Instead compare to a data range of interest.

Note that the relational operators are either binary; they take only two values. The following is not a valid way to see if \( n \) is between 2 and 5:

\[ 2 < n < 5 \]

The accepted way is to use Boolean operators.

**Boolean Operators**

A simple Boolean expression is either a Boolean variable or constant or an expression involving the relational operators that evaluates to either true or false. These simple Boolean expressions can be combined using the logical operations defined on Boolean values. There are three Boolean operators: AND, OR, and NOT. Here is a table showing the meaning of these operators and the symbols that are used to represent them in VB.NET.

<table>
<thead>
<tr>
<th>VB.NET Keyword</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>AND is a binary Boolean operator. If both operands are true, the result is true. Otherwise, the result is false.</td>
</tr>
<tr>
<td></td>
<td><strong>True</strong></td>
</tr>
<tr>
<td></td>
<td><strong>True</strong></td>
</tr>
<tr>
<td></td>
<td><strong>False</strong></td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>OR is a binary Boolean operator. If at least one of the operands is true, the result is true. Otherwise, the result is false.</td>
</tr>
<tr>
<td></td>
<td><strong>True</strong></td>
</tr>
<tr>
<td></td>
<td><strong>True</strong></td>
</tr>
<tr>
<td></td>
<td><strong>False</strong></td>
</tr>
</tbody>
</table>
NOT is a unary Boolean operator. NOT changes the value of its operand: If the operand is true, the result is false; if the operand is false, the result is true.

<table>
<thead>
<tr>
<th>Not_Value</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

If relational operators and Boolean operators are combined in the same expression the Boolean operator NOT has the highest precedence, the relational operators have next higher precedence, and the Boolean operators AND and OR come last (in that order). Expressions in parentheses are always evaluated first.

For example, given the following expression (stop is a boolean variable)

\[
\text{count} \leq 10 \text{ and } \text{sum} \geq \text{limit} \text{ or not stop}
\]

**not stop** is evaluated first, the expressions involving the relational operators are evaluated next, the AND is applied, and finally the OR is applied.

It is a good idea to use parenthesis to make your expressions more readable, e.g:

\[
((\text{count} \leq 10) \text{ AND } (\text{sum} \geq \text{limit})) \text{ OR } (\text{NOT (stop}))
\]

This also helps avoid difficult-to-find errors if the programmer forgets the precedence rules.

A common error is to replace the condition
\[
\text{Not (} 2 < 3 \text{ }\)
\]
by the condition
\[
(2 > 3)
\]
The correct replacement is \(2 \geq 3\) because \(\geq\) is the opposite of \(<\), just as \(\leq\) is the opposite of \(>\).
Exercises: Are the following statements true or false?

- Dim a as Integer = 2
- Dim b as Integer = 3
- 3*a = 2*b
- (a<b) Or (b<a)
- (2<a) And (a<5)
- Not ((a<b) And (a<(b+a)))
- ((a=b) And (a*<b+b)) Or ((b<a) And (2*a<b))
- “Car”<“Train”
- “99”>”ninety-nine”
- “9W” > “9a”
- (“Ant” < “hill”) And (“mole” > “hill”) Or Not (Not (“Ant” < “hill”) Or Not (“Mole” > “hill”))

If-Then and If-Then-Else Statements

The If statement allows the programmer to change the logical order of a program; that is, make the order in which the statements are executed differ from the order in which they are listed in the program. The If-Then statement uses a Boolean expression to determine whether to execute a statement or to skip it. The format is as follows:

```
If (boolean_expression1)
  statement1   ' Expr1 true
  ElseIf (boolean_expression2)
    statement2   ' Expr1 false, Expr2 true
  ElseIf (boolean_expression3)
    statement3   ' Expr1 false, Expr2 false, Expr3 true
  ...
  Else
    statement_all_above_false   ' Expr1, Expr2, Expr3 false
End If
```

The Else and ElseIf portions are optional. If you like you can leave them off. You can also insert multiple statements into each section if you have more than one line of code you would like to execute for each block.
Here are some examples of if statements.

To find the larger of two numbers:

```vba
Dim num1, num2, largerNum As Double
num1 = CDbl(txtFirstNum.Text)
num2 = CDbl(txtSecondNum.Text)
If num1 > num2 Then
    largerNum = num1
Else
    largerNum = num2
End If
txtResult.Text = "The larger number is " & largerNum
```

Setting an appropriate message for profit/loss:

```vba
If costs = revenue Then
    txtResult.Text = "Break even"
Else
    If costs < revenue Then
        profit = revenue - costs
        txtResult.Text = "Profit is " & FormatCurrency(profit)
    Else
        loss = costs - revenue
        txtResult.Text = "Loss is " & FormatCurrency(loss)
    End If
End If
```

Checking an answer for how much a ten gallon hat holds:

```vba
Dim answer As Double
answer = CDbl(txtAnswer.Text)
If (answer >= 0.5) And (answer <= 1) Then
    txtSolution.Text = "Good, "
Else
    txtSolution.Text = "No, "
End If
txtSolution.Text &= "it holds about 3/4 of" & " a gallon."
```

Code that takes as input a number between 0-100 and outputs a letter grade, where 90-100 is A, 80-90 is B, 70-80 is C, 60-70 is D, and anything below 60 is an F.
numGrade = CDbl(textBoxGrade.Text)
If (numGrade >= 90) Then
    Console.WriteLine("A")
ElseIf (numGrade >= 80) Then
    Console.WriteLine("B")
ElseIf (numGrade >= 70) Then
    Console.WriteLine("C")
ElseIf (numGrade >= 60) Then
    Console.WriteLine("D")
Else
    Console.WriteLine("F")
End If

Note that anything can go inside the body of the If statement – including other If statements! When we do this, it is called nested If statements. For example:

If (numGrade >= 90) Then
    If (numGrade > 96) Then
        Return "A+
    ElseIf (numGrade > 93) Then
        Return "A"
    Else
        Return "A-
    End If
ElseIf (numGrade >= 80) Then
    Return "B"
End If

In general, any Nested If statement can be turned into a single If statement using AND’s as follows:

If cond1 Then If cond1 And cond2 Then
    If cond2 Then action
    action
    End If
End If
End If

The format on the right is generally less confusing, although there are exceptions.
In-Class Exercise: Write a program that gives a short quiz about UAA (2004 data):

1. Who is the current Dean of the College of Arts & Sciences?
   a. Kerry Feldman
   b. Jim Liszka
   c. Theodore Kassier

2. This is the most popular major at UAA, in terms of number of students enrolled in the program.
   a. Biological Sciences
   b. Management
   c. Nursing

3. Graduation rate for first time, full time baccalaureate students at UAA
   a. 57%
   b. 69%
   c. 75%

At the end of the quiz, display the score of the test-taker, where 1 point is awarded for each correct question.

In-Class Exercise: Write a program that takes a year and determines if it is a leap year. Every year divisible by four is a leap year, with the exception of years divisible by 100 and not divisible by 400. For example:

1600 is a leap year: Divisible by 4, Divisible by 100, and Divisible by 400
2000 is a leap year: Divisible by 4, Divisible by 100, and Divisible by 400
1984 is a leap year: Divisible by 4, Not divisible by 100, Not divisible by 400
1700 is not a leap year: Divisible by 4, Divisible by 100, but not divisible by 400

Select Case Blocks

A Select Case block a more compact way to construct what is equivalent to an if-then-elseif statement. Select statements use the value of a single expression called the selector. Possible actions are executed depending on the value of the selector.

The general format of a select block is:

```
Select Case selector
   Case valueList1
      action1
   Case valueList2
      action2
   Case Else
      action of last resort
End Select
```

The Case Else is optional.
Here is an example of using a select statement to take the finishing position of a horse and indicate if the outcome is Win, Place, or Show:

```
Dim position As Integer
position = CInt(txtPosition.Text)
Select Case position 'position is the selector
    Case 1
        txtOutcome.Text = "Win"
    Case 2
        txtOutcome.Text = "Place"
    Case 3
        txtOutcome.Text = "Show"
    Case 4, 5
        txtOutcome.Text = "You almost placed in the money."
    Case Else
        txtOutcome.Text = "Out of the money."
End Select
```

Note that we can insert lists of values. For example, 4 and 5 are separated by a comma. If the user entered either 4 or 5 then txtOutcome.Text would be set to “You almost placed in the money.”

We can specify data ranges and also use relational operators as show below:

```
Dim position As Integer
position = CInt(txtPosition.Text)
Select Case position
    Case 1 To 3
        txtOutcome.Text = "In the money."
    Case Is >= 4
        txtOutcome.Text = "Not in the money."
End Select
```

**In-Class Exercise:**

Write a program that inputs a number between 0 and 99 and outputs the value in English, e.g. “ninety nine” for 99, “zero” for 0, etc. Don’t use 100 different WriteLine statements!