STARTING OUT WITH

Visual Basic® 2010

fifth edition

Tony Gaddis

Kip Irvine
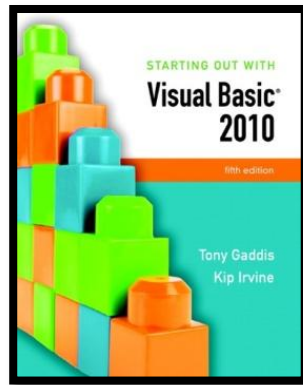
Chapter 4

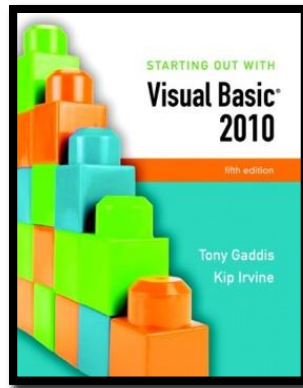## Making Decisions

Introduction

- This chapter covers the Visual Basic decision statements
  - **If...Then**
  - **If...Then...Else**
  - **If...Then...ElseIf**
  - **Select Case**
- It also discusses the use of
  - Radio Buttons
  - Check Boxes
  - Message Boxes
  - Input Validation

THE DECISION STRUCTURE

The decision structure allows a program's logic to have more than one path of execution.

Order of Statement Execution

- Thus far, our code has been executed sequentially in a sequence structure

- To write meaningful programs we need multiple paths of execution
  - Some statements should be executed under certain circumstances in a decision structure
  - This chapter presents the means to execute statements conditionally
  - Next chapter presents the means to execute the same statements repeatedly

The Decision Structure

- Flowchart of a typical decision structure
- Evaluate the condition
  - Is it cold outside?
- Execute or skip over some code
  - If yes, wear a coat

Is it cold outside?

True

False

Wear a coat.
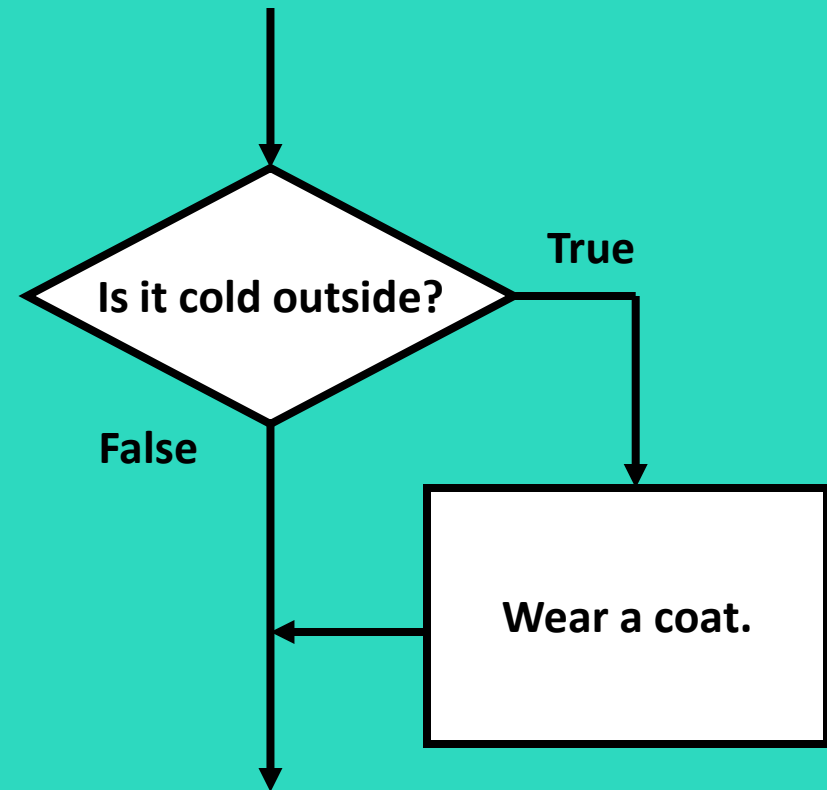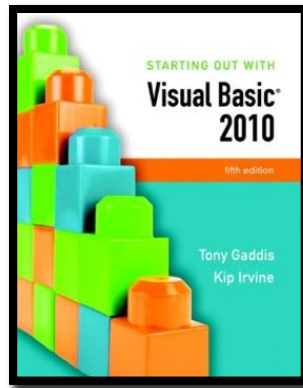
Section 4.2

THE IF...THEN STATEMENT

The If...Then statement causes other statements to execute only when an expression is true.

General Format

If *expression* Then
   statement
   (more statements may follow)
End If

- If the **expression** is **True**, execute the statements between **If...Then** and **End If**
- Otherwise, the statements are skipped

Relational Operators

- Usually a condition is formed using a relational operator
- A relational operator determines if a specific relationship exists between two values

>	Greater than
<	Less than
=	Equal to
<>	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

Boolean Expressions

- Relational operators are binary – meaning they use two operands, for example:

  length > width   **Is length greater than width?**

  size <= 10   **Is size less than or equal 10?**

- Relational operators are used in Boolean expressions which yield a true or false result

Putting It All Together

If...Then statement examples:

```
If decSales > 50000 Then
    MessageBox.Show("You've earned a bonus!")
End If


If decSales > 50000 Then
    MessageBox.Show("You've earned a bonus!")
    decCommissionRate = 0.12
    intDaysOff = intDaysOff + 1
End If
```

Rules to Remember

- The **If** and the **Then** must be on the same line
- Only a remark may follow the **Then**
- The **End If** must be on a separate line
- Only a remark may follow the **End If**

- Tutorial 4-1 presents an application that uses the **If…Then** statement

Programming Style

- The code between the **If…Then** and the **End If** is indented

- Visual Basic does not require this

- It is a convention among programmers to aid in the readability of programs

- By default, the Visual Basic editor will automatically do this indentation as you enter your program

Using Relational Operators with Math Operators

- Math operators are evaluated before relational operators

  If intX + intY > intA - intB Then
  lblMessage.Text = "It is true!"
  End If

- **intX + intY** and **intA - intB** are evaluated first

- Most programmers prefer to use parentheses to clarify the order of operations

  If (intX + intY) > (intA – intB) Then
  lblMessage.Text = "It is true!"
  End If

Using Function Calls with Relational Operators

- Either or both relational operator operands may be function calls

```
If CInt(txtInput.Text) < 100 Then
        lblMessage.Text = "It is true!"
End If
```

- The return value of the function call is compared to the value using the relational operator

Using Boolean Variables as Flags

- A flag is a Boolean variable that signals when some condition exists in the program

- Since a Boolean variable is either **True** or **False**, it can be used as the condition of an **If...Then** statement
  - Since a Boolean variable already evaluates to **True** or **False**, an **=** operator is not required

```
If blnQuotaMet Then
        lblMessage.Text = "You have met your sales quota"
End If
```
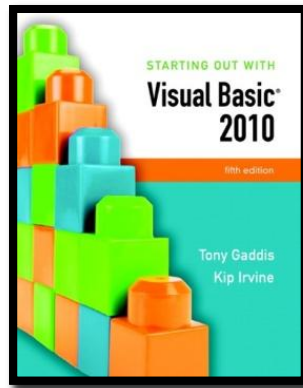
Section 4.3

THE IF…THEN…ELSE STATEMENT

The If…Then…Else statement executes one group of statements if the Boolean expression is true and another group of statements if the Boolean expression is false.
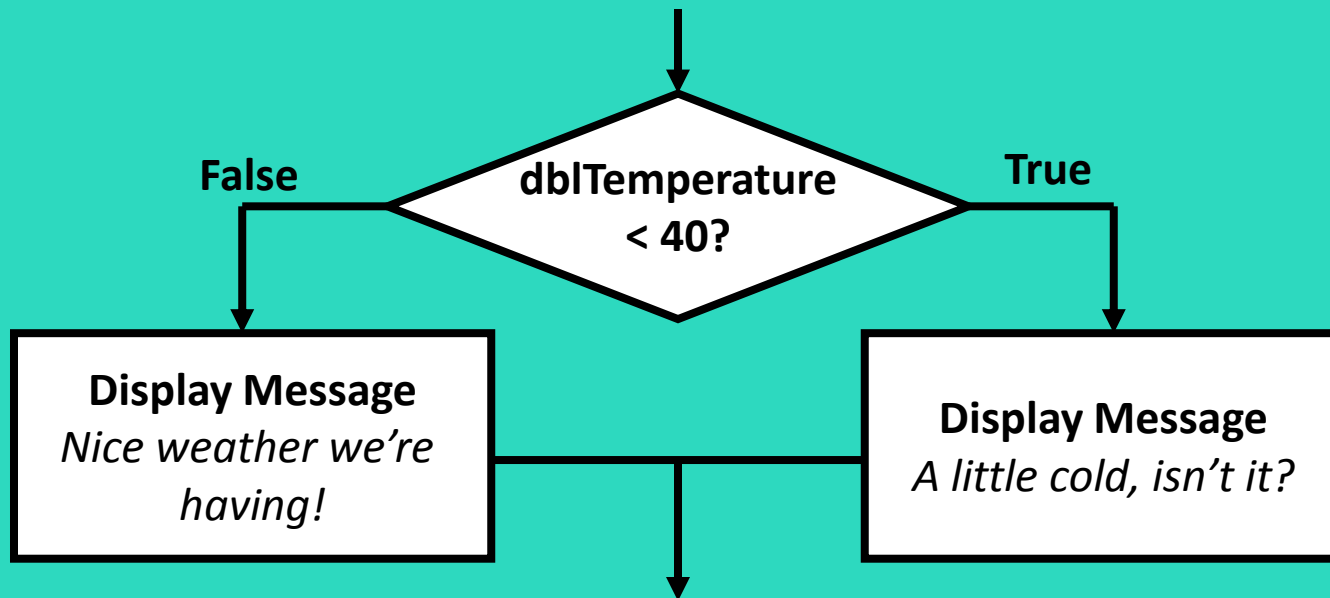
Addison Wesley
is an imprint of

General Format

```
If expression Then
    statement
    (more statements may follow)
Else
    statement
    (more statements may follow)
End If
```

- If the **expression** is **True**
  - execute the statements between **If...Then** and **Else**
- If the **expression** is **False**
  - execute the statements between **Else** and **End If**

Flowchart and Pseudocode

False dblTemperature < 40? **True**

Display Message
*Nice weather we're having!*

Display Message
A little cold, isn't it?

If temperature < 40 Then
  Display the message "A little cold, isn't it?"
Else
  Display the message "Nice weather we're having!"
End If

Two Mutually Exclusive Choices

- The **If...Then...Else** has two choices
  - The condition will either be True or False
  - So either the **Then** clause or **Else** clause will be executed
  - These are two mutually exclusive choices
- Tutorial 4-2 contains an example of the **If...Then...Else** construct
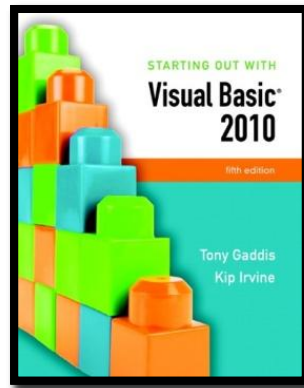
THE IF...THEN...ELSEIF STATEMENT

The If...Then...ElseIf statement is like a chain of
If...Then...Else statements. They perform their tests,
one after the other, until one of them is found to be true.

Multiple Possible Choices

- The **If…Then…ElseIf** statement allows for an entire series of possible choices

- In pseudocode:

> *If it is very cold Then*
> > *Wear a coat*
>
> *Elseif it is chilly*
> > *Wear a light jacket*
>
> *Elseif it is windy*
> > *Wear a windbreaker*
>
> *Elseif it is hot*
> > *Wear no jacket*

Multiple Possible Choices

- Each of the series of conditions in an **If…Then…ElseIf** is tested in sequence
- When a condition is true, the remaining conditions are ignored
- The order of the conditions is vital
  - Wrong order can result in wrong decision - called a logic error
  - What if it's chilly *and* windy?
  - If windy is tested before chilly, you'd go out with a windbreaker when you need a jacket

General Format

If *expression* Then
   statement
   (more statements may follow)
ElseIf *expression* Then
   statement
   (more statements may follow)
(put as many ElseIf statements as necessary)
Else
   statement
   (more statements may follow)

- This construction is like a chain of **If...Then...Else** statements
- The **Else** part of one statement is linked to the **If** part of another

Flowchart
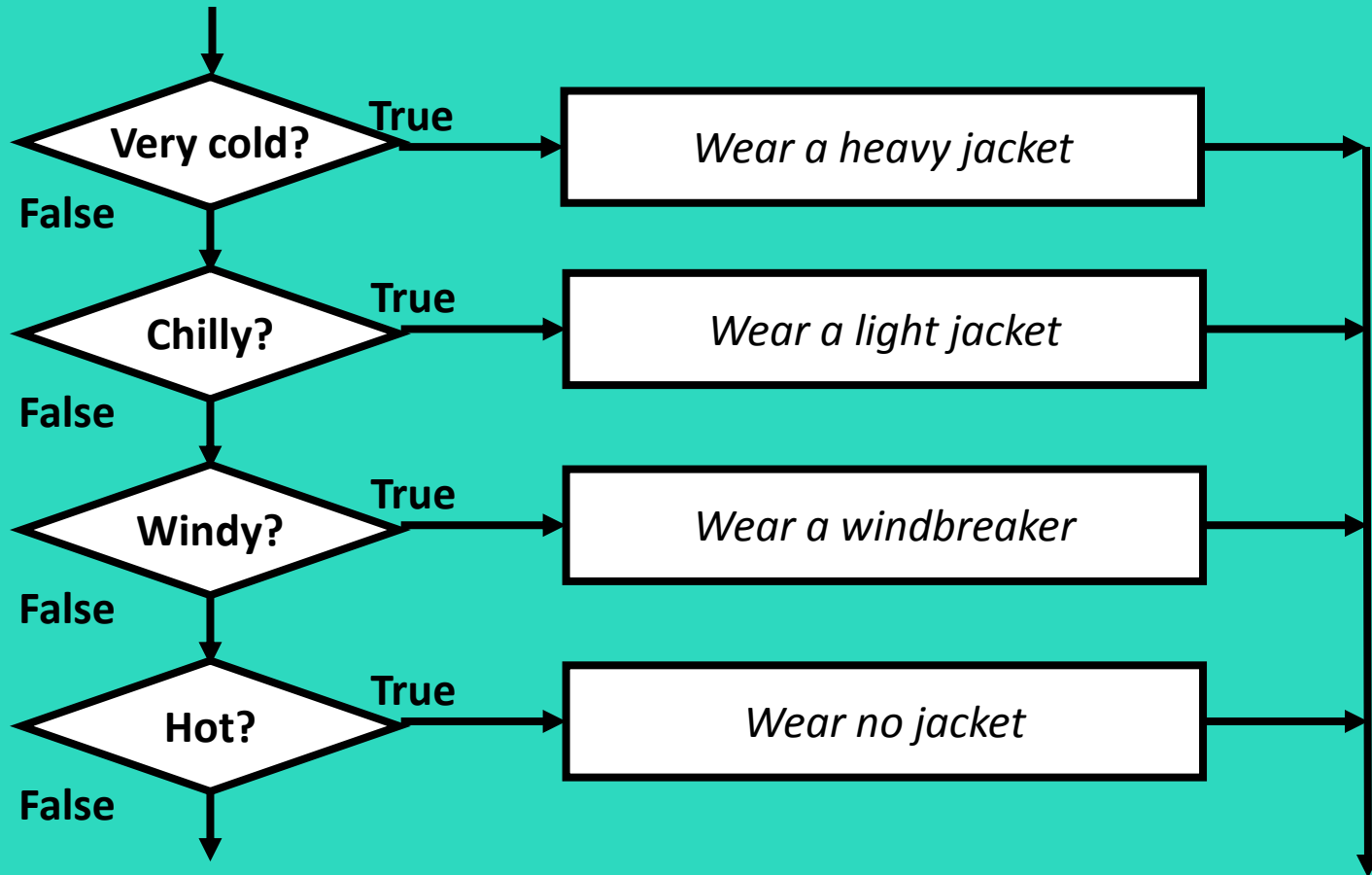
Example of ElseIf Usage

```
If dblAverage < 60 Then
        lblGrade.Text = "F"
ElseIf dblAverage < 70 Then
        lblGrade.Text = "D"
ElseIf dblAverage < 80 Then
        lblGrade.Text = "C"
ElseIf dblAverage < 90 Then
        lblGrade.Text = "B"
ElseIf sngAverage <= 100 Then
        lblGrade.Text = "A"
End If
```

- Does the order of these conditions matter?
- What happens if we reverse the order?

Using Only If...Then Statements
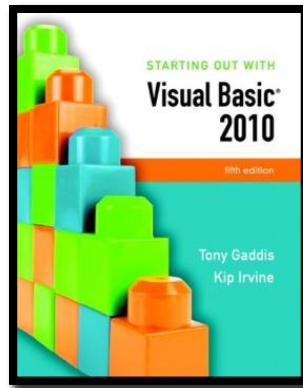
```
If dblAverage < 60 Then
        lblGrade.Text = "F"
End If
If dblAverage < 70 Then
        lblGrade.Text = "D"
End If
If dblAverage < 80 Then
        lblGrade.Text = "C"
End If
If dblAverage < 90 Then
        lblGrade.Text = "B"
End If
If dblAverage <= 100 Then
        lblGrade.Text = "A"
End If
```

- Does this code function correctly?
- What is assigned to lblGrade for a 65 average? 75?

Using a Trailing Else

- A sequence of **ElseIf** statements may end with a plain **Else**, called a trailing Else

- If none of the conditions are **True**, the trailing **Else** statement(s) will be executed

- The trailing **Else** catches any value that falls through the cracks

```
' Display the letter grade.
If dblAverage < 60 Then
lblGrade.Text = "F"
ElseIf dblAverage < 70 Then
lblGrade.Text = "D"
ElseIf dblAverage < 80 Then
lblGrade.Text = "C"
ElseIf dblAverage < 90 Then
lblGrade.Text = "B"
ElseIf dblAverage <= 100 Then
lblGrade.Text = "A"
Else
lblGrade.Text = "Invalid Score"
End If
```

NESTED IF STATEMENTS

A nested If statement is an If statement in the conditionally executed code of another If statement. (In this section, we use the term If statement to refer to an If . . . Then, If...Then...Else, or If...Then...ElseIf statement.)

If Statements Within If Statements

- Any type of statement may be used inside a set of **Then**, **Else**, or **ElseIf** statements of an **If**

- This includes other **If** statements

- **If** statements within **If** statements create a more complex decision structure called a Nested If
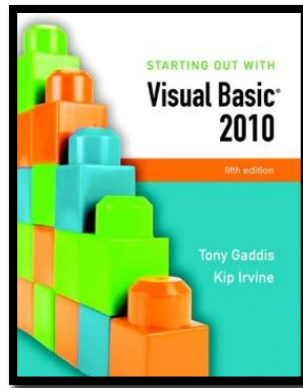
Nested If Example

- Tutorial 4-4 examines an application that uses nested **If** Statements

- In the application, the customer must meet one of the following qualifications:

    - Earn $30,000 per year or more and have worked in his or her current job for more than two years.

    - Have worked at his or her current job for more than five years.

Examining the Nested If Statement

```
If dblSalary > 30000 Then
        If intYearsOnJob > 2 Then
                lblMessage.Text = "Applicant qualifies."
        Else
                lblMessage.Text = "Applicant does not qualify."
        End If
Else
        If intYearsOnJob > 5 Then
                lblMessage.Text = "Applicant qualifies."
        Else
                lblMessage.Text = "Applicant does not qualify."
        End If
End If
```

Flowchart of Nested If Statements
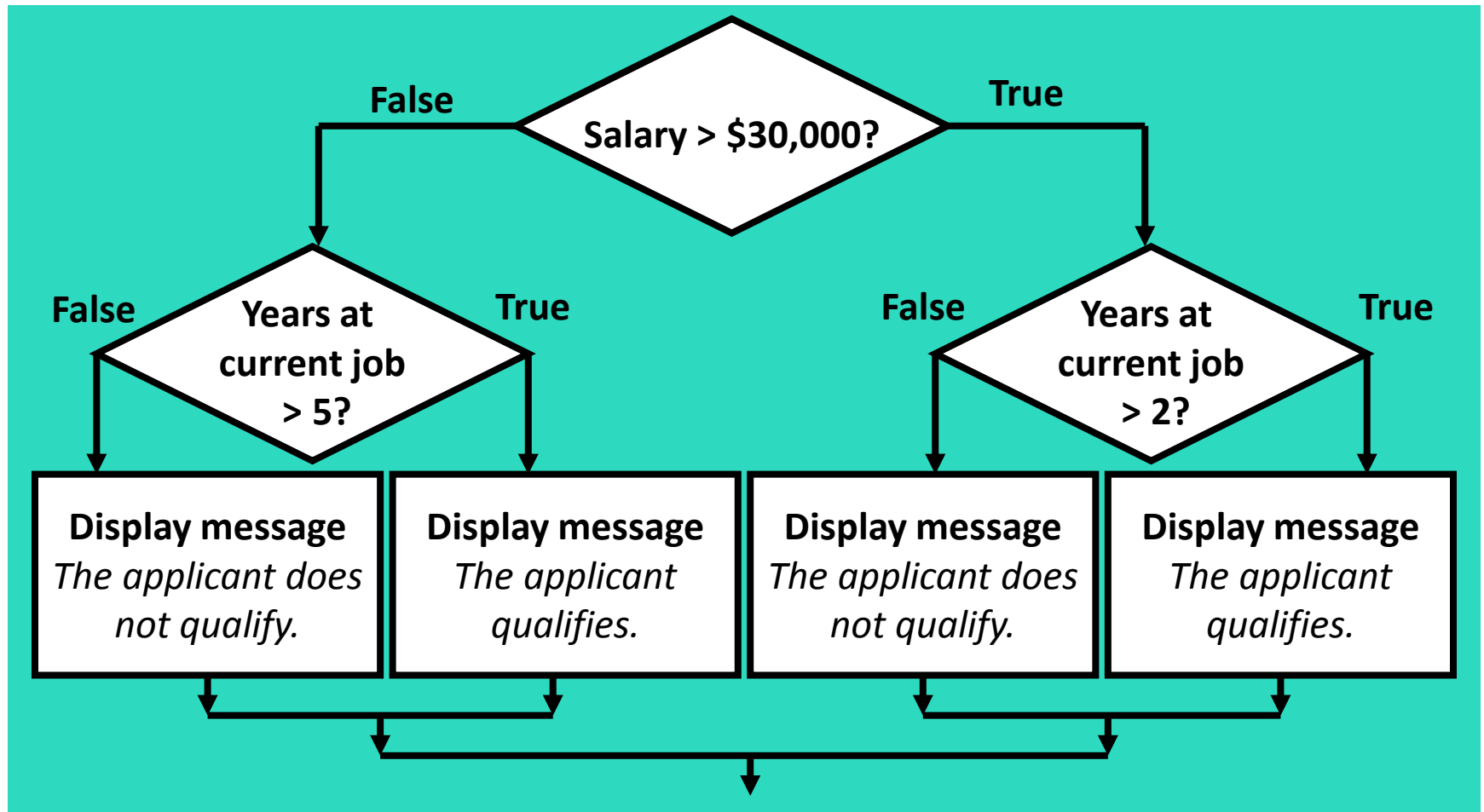
Section 4.6

LOGICAL OPERATORS

Logical operators combine two or more Boolean expressions into a single expression.

Visual Basic Logical Operators

- Visual Basic provides Logical operators that can combine multiple Boolean expressions into a compound expression

Operator	Effect
And	Combines two expressions into one. Both expressions must be true for the overall expression to be true.
Or	Combines two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.
Xor	Combines two expressions into one. One expression (not both) must be true for the overall expression to be true. If both expressions are true, or both expressions are false, the overall expression is false.
Not	Reverses the logical value of an expression: makes a true expression false and a false expression true.

The And Operator

- The **And** operator combines two expressions into one

- The following **If** statement uses the **And** operator:

  If intTemperature < 20 And intMinutes > 12 Then
  lblMessage.Text = "The temperature is in the danger zone."
  End If

- Both expressions must be true for the overall expression to be true, as shown in the following truth table:

Expression 1	Expression 2	Expression 1 And Expression 2
True	False	False
False	True	False
False	False	False
True	True	True

Short-Circuit Evaluation with AndAlso

- When using the **And** operator, if the first expression is false, then the entire expression will be false

- If there is no need to evaluate the second expression, it can be skipped using a method called short-circuit evaluation

- In Visual Basic you use the **AndAlso** operator to achieve short-circuit evaluation

Short-Circuit Evaluation with AndAlso

- In the following example, assuming that **dblX** is less than or equal to zero, **CheckValue** is not called and *Expression is False is displayed:*

```
If dblX > 0 AndAlso CheckValue(dblX) Then
    lblResult.Text = "Expression is True"
Else
    lblResult.Text = "Expression is False"
End If
```

The Or Operator

- The **Or** operator combines two expressions into one

- The following **If** statement uses the **Or** operator:

  If intTemperature < 20 Or intTemperature > 100 Then
     lblMessage.Text = "The temperature is in the danger zone."
  End If

- One or both expressions must be true for the overall expression to be true, as shown in the following truth table:

Expression 1	Expression 2	Expression 1 Or Expression 2
True	False	True
False	True	True
False	False	False
True	True	True

Short Circuit-Evaluation with OrElse

- When using the **Or** operator, if the first expression is true, then the entire expression will be true

- If there is no need to evaluate the second expression, it can be skipped using short-circuit evaluation with the **OrElse** operator

Short Circuit-Evaluation with OrElse

- In the following example, if **dblX** is equal to zero, **CheckValue** is not called and *Expression is True is displayed:*

  ```
  If dblX = 0 OrElse CheckValue(dblX) Then
  lblResult.Text = "Expression is True"
  End If
  ```

The Xor Operator

- The **Xor** operator combines two expressions into one

- **Xor** stands for *exclusive or*

- The following **If** statement uses the **Xor** operator:

      If decTotal > 1000 Xor decAverage > 120 Then
      lblMessage.Text = "You may try again."
      End If

- One but not both expressions must be true for the overall expression to be true, as shown in the following truth table:

Expression 1	Expression 2	Expression 1 Xor Expression 2
True	False	True
False	True	True
False	False	False
True	True	False

The Not Operator

- The **Not** operator takes a Boolean expression and reverses its logical value

- The following **If** statement uses the **Not** operator:

  If Not intTemperature > 100 Then
  lblMessage.Text = "You are below the maximum temperature."
  End If

- If the expression is true, the **Not** operator returns **False**, and if the expression is false, it returns **True**, as shown in the following truth table:

Expression	Not Expression
True	False
False	True

Checking Numerical Ranges

- The **And** operator is best for checking if a value is *inside* a range of numbers

  ```
  If intX >= 20 And intX <= 40 Then
  lblMessage.Text = "The value is in the acceptable range."
  End If
  ```

- The **Or** operator is best for checking if a value is *outside* a range of numbers

  ```
  If intX < 20 Or intX > 40 Then
  lblMessage.Text = "The value is outside the acceptable range."
  End If
  ```

Precedence of Logical Operators

- Logical operators have an order of precedence just as arithmetic operators do
- From highest to lowest precedence
  - **Not**
  - **And**
  - **Or**
  - **Xor**
- As with arithmetic operations, parentheses are often used to clarify order of operations

Precedence of Logical Operators

- For example, in the statement

    If x < 0 And y > 100 Or z = 50

    - **x < 0 And y > 100** is evaluated first
    - If the **And** condition is true, we then evaluate
    - **True Or z = 50**
    - If the **And** condition is false, we then evaluate
    - **False Or z = 50**

- If the **Or** condition is to be evaluated first, parentheses must be used

    If x < 0 And (y > 100 Or z = 50)

Math, Relational, & Logical Operators

- Evaluate the following if: **a=5, b=7, x=100, y=30**

  If x > a * 10 And y < b + 20

  Evaluating the math operators leaves us with

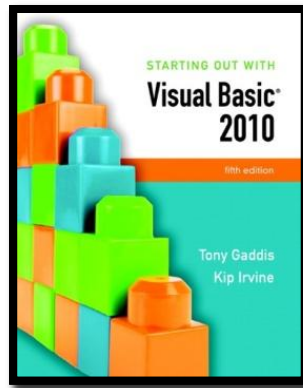 **If x > 50 And y < 27**

  Evaluating the relational operators leaves

  If True And False

  Evaluating the logical operators leaves

  False

- Parentheses make order of operations clear

  If (x > (a * 10)) And (y < (b + 20))

Section 4.7

COMPARING, TESTING, AND WORKING WITH STRINGS

Visual Basic provides various methods in the String class that make it easy to work with strings. This section shows you how to use relational operators to compare strings, and discusses several functions and string methods that perform tests and manipulations on strings.

Addison Wesley
is an imprint of

Strings Can Be Compared

- Relational operators can be used to compare strings and string literals

```
strName1 = "Mary"
strName2 = "Mark"
If strName1 = strName2 Then
        lblMessage.Text = " Names are the same"
Else
        lblMessage.Text = " Names are NOT the same"
End If

If strMonth <> "October" Then
        ' statement
End If
```

How Are Strings Compared?

- Characters are stored as numeric values
- Visual Basic uses Unicode
- The Unicode numbering system represents:
  - All letters of the alphabet
  - Printable digits 0 through 9
  - Punctuation symbols and special characters
- Letters (A,B,C) are arranged alphabetically
  - The numeric value of **A** is less than the numeric value of **B**

- Characters of each string are compared one by one until a difference is found
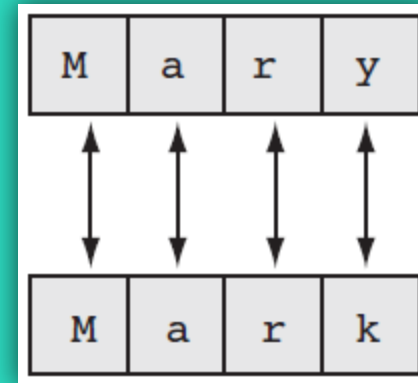
M	a	r	y
M	a	r	k

Mary is greater than Mark because **"y"** has a Unicode value greater than **"k"**

Testing for No Input

- The predefined constant **String.Empty** represents an empty string, which is a string that contains no characters

  If txtInput.Text = String.Empty Then
  lblMessage.Text = "Please enter a value"
  Else
  ' The txtInput control contains input, so
  ' perform an operation with it here.
  End If

- Useful for determining whether the user has provided input for a required field before performing operations on that field

The ToUpper and ToLower Methods

- The **ToUpper** method can be applied to a string

- Results in a string with lowercase letters converted to uppercase

- The original string is not changed

- General Format:

  StringExpression.ToUpper()

- In the following example, **strBigWord** is assigned the string **"HELLO"** using the **ToUpper** method:

  strLittleWord = "Hello"
  strBigWord = strLittleWord.ToUpper()

- The **ToLower** method can be applied to a string

- Results in a string with uppercase letters converted to lowercase

- The original string is not changed

- General Format:

  StringExpression.ToLower()

- In the following example, **strLittleTown** is assigned the string **"new york"** using the **ToLower** method:

  strBigTown = "NEW YORK"
  strLittleTown = strBigTown.ToLower()

A Handy Use for ToUpper or ToLower

- **ToUpper** or **ToLower** can be used to perform case insensitive comparisons of strings
- 1st comparison below is false **"HELLO" <> "hello"**
- 2nd comparison is true
- **ToLower** converts both strings to lower case
- Causes **"hello"** to be compared to **"hello"**

```
strWord1 = "HELLO"
strWord2 = "hello"
If strWord1 = strWord2                          ' False, not equal
If strWord1.ToLower() = strWord2.ToLower()      ' True, equal
```

- Tutorial 4-5 demonstrates how this is used

The IsNumeric Function

- This function accepts a string as an argument and returns **True** if the string contains a number

  ```
  Dim strNumber As String
  strNumber = "576"
  If IsNumeric(strNumber)          ' Returns true
  strNumber = "123abc"
  If IsNumeric(strNumber)          ' Returns false
  ```

- Use **IsNumeric** function to determine if a given string contains numeric data

Determining the Length of a String

- The Length property, a member of the **String** class, returns the number of characters in a string

- In the following example, the **intNumChars** variable contains the value **6**:

```
Dim strName As String = "Herman"
Dim intNumChars As Integer
intNumChars = strName.Length
```

- You can also determine the length of a control's **Text** property, as shown in the following code:

```
If txtInput.Text.Length > 20 Then
lblMessage.Text = "Please enter no more than 20 characters."
End If
```

Trimming Spaces from Strings

- There are three methods that remove spaces from strings:
  - **TrimStart** : removes leading spaces
  - **TrimEnd**   : removes trailing spaces
  - **Trim**   : removes leading and trailing spaces
- Here is the general format for each method:

   StringExpression.TrimStart()
   StringExpression.TrimEnd()
   StringExpression.Trim()

- An example with three leading and trailing spaces, using each method:

  strGreeting = "   Hello   "
  lblMessage1.Text = strGreeting.TrimStart() ' lblMessage1.Text = "Hello   "
  lblMessage2.Text = strGreeting.TrimEnd()   ' lblMessage2.Text = "   Hello"
  lblMessage3.Text = strGreeting.Trim()   ' lblMessage3.Text = "Hello"

The Substring Method

- The **Substring** method returns a portion of a string or a *"string within a string"* (a substring)

- Each character position is numbered sequentially with the 1st character referred to as position zero

- ***StringExpression.Substring(Start)***
  - Returns the characters from the **Start** position to the end of the string

- ***StringExpression.Substring(Start, Length)***
  - Returns the number of characters specified by **Length** beginning with the **Start** position

Substring Method Examples

- The first example starts at the 8th (**W**) character in the string and continues to the end of the string:

  ```
  Dim strLastName As String
  Dim strFullName As String = "George Washington"
  strLastName = strFullName.Substring(7)        ' Washington
  ```

- The second example starts at the beginning (**G**) of the string and continues until it reaches the 7th (empty space) character of the string:

  ```
  Dim strFirstName As String
  Dim strFullName As String = "George Washington"
  strFirstName = strFullName.Substring(0, 6)    ' George
  ```

The IndexOf Method

- The **IndexOf** method searches for a character or string within a string, it has three general formats:

- *StringExpression.IndexOf(Searchstring)*

  - Searches the entire string for *SearchString*

- *StringExpression.IndexOf(SearchString, Start)*

  - Starts at the character position *Start* and searches for *SearchString* from that point

- *StringExpr.IndexOf(SearchString, Start, Count)*

  - Starts at the character position *Start* and searches *Count* characters for *SearchString*

IndexOf Method Examples

- The **IndexOf** method will return the starting position of the **SearchString** in the string being searched
- Positions are numbered from **0** (for the first)
- If **SearchString** is not found, a value of **-1** is returned

Position 0	Position 9

```
Dim name As String = "Angelina Adams"
Dim position As Integer
position = name.IndexOf("A", 1)
            ' position has the value 9
```

- Tutorial 4-6 provides an opportunity to work with several of the string methods

Section 4.8

MORE ABOUT MESSAGE BOXES

Sometimes you need a convenient way to display a message to the user. This section discusses the **MessageBox.Show** method, which allows you to display a message in a dialog box.

Addison Wesley
is an imprint of

PEARSON

Message Box Arguments

- A message box is a dialog box with a user message in a pop-up window

  MessageBox.Show(Message, Caption, Buttons, Icon, DefaultButton)

- The following can be specified
  - ***Message*** - text to display within the box
  - ***Caption*** - title for the top bar of the box
  - ***Buttons*** - indicates which buttons to display
  - ***Icon*** - indicates icon to display
  - ***DefaultButton*** - indicates which button corresponds to the Return Key
- ***Message*** is required, the remaining arguments are optional
- Use of an argument requires those before it

The Optional Buttons Argument

- Unless specified, the message box has only an *OK button*

- ***Buttons** is* a value that specifies which buttons to display in the message box

MessageBox.Show(Message, Caption, Buttons, Icon, DefaultButton)

Value	Description
MessageBoxButtons.AbortRetryIgnore	Displays *Abort, Retry, and Ignore buttons*
MessageBoxButtons.OK	Displays only an *OK button*
MessageBoxButtons.OKCancel	Displays *OK and Cancel buttons*
MessageBoxButtons.RetryCancel	Displays *Retry and Cancel buttons*
MessageBoxButtons.YesNo	Displays *Yes and No buttons*
MessageBoxButtons.YesNoCancel	Displays *Yes, No, and Cancel buttons*

The Optional Icon Argument

- ***Icon** is a value that specifies an icon to display* in the message box

  MessageBox.Show(Message, Caption, Buttons, Icon, DefaultButton)

- For example:

  MessageBox.Show("Do you wish to continue?", "Please Confirm", MessageBoxButtons.YesNo, MessageBoxIcon.Question)
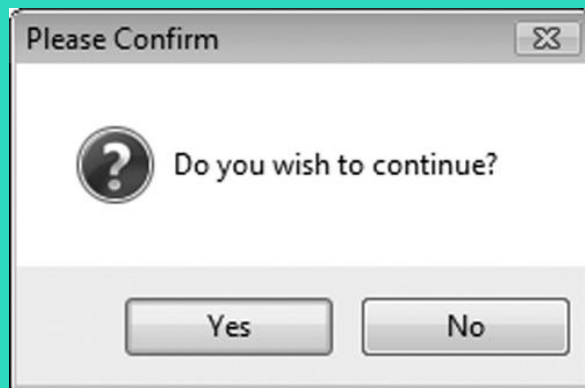
Value	Image
MessageBoxIcon.Asterisk **MessageBoxIcon.Information**	
MessageBoxIcon.Error **MessageBoxIcon.Hand** **MessageBoxIcon.Stop**	
MessageBoxIcon.Exclamation **MessageBoxIcon.Warning**	
MessageBoxIcon.Question	

The Optional DefaultButton Argument

- The **DefaultButton** *argument specifies which button* to select as the default button

- The default button is the button clicked when the user presses the Enter key

  MessageBox.Show(Message, Caption, Buttons, Icon, *DefaultButton*)

Value	Description
MessageBoxDefaultButton.Button1	Selects the leftmost button on the message box as the default button
MessageBoxDefaultButton.Button2	Selects the second button from the left edge of the message box as the default button
MessageBoxDefaultButton.Button3	Selects the third button from the left edge of the message box as the default button

MessageBox Example

- The following statement displays a message box and selects Button2 (the *No* button) as the default button:

  MessageBox.Show("Do you wish to continue?",
  "Please Confirm",
  MessageBoxButtons.YesNo,
  MessageBoxIcon.Question,
  MessageBoxDefaultButton.Button2)
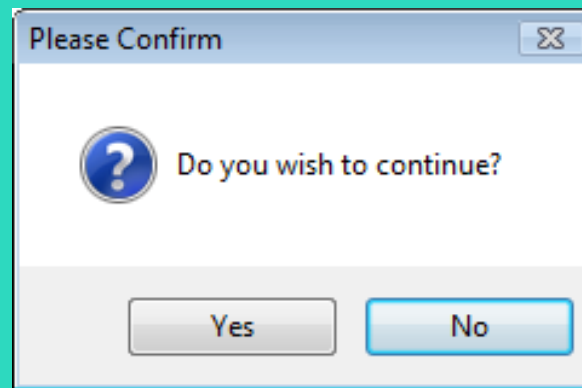
Determining Which Button the User Clicked

- The **MessageBox.Show** method returns an integer that indicates which button the user clicked

Value	Meaning
Windows.Forms.DialogResult.Abort	The user clicked the *Abort button*
Windows.Forms.DialogResult.Cancel	The user clicked the *Cancel button*
Windows.Forms.DialogResult.Ignore	The user clicked the *Ignore button*
Windows.Forms.DialogResult.No	The user clicked the *No button*
Windows.Forms.DialogResult.OK	The user clicked the *OK button*
Windows.Forms.DialogResult.Retry	The user clicked the *Retry button*
Windows.Forms.DialogResult.Yes	The user clicked the *Yes button*

Determining Which Button the User Clicked Example Code

- The following code shows how an **If** statement can take actions based on which message box button the user clicked:

```
Dim intResult As Integer
intResult = MessageBox.Show("Do you wish to continue?",
                            "Please Confirm",
                            MessageBoxButtons.YesNo)
If intResult = Windows.Forms.DialogResult.Yes Then
    ' Perform an action here
ElseIf intResult = Windows.Forms.DialogResult.No Then
    ' Perform another action here
End If
```

Using **ControlChars.CrLf** to Display Multiple Lines

- If you want to display multiple lines of information in a message box, use the constant **ControlChars.CrLf**
  - **CrLf** stands for *Carriage return Line feed*
  - Concatenate(**&**) it with the string you wish to display, where you wish to begin a new line

    MessageBox.Show("This is line 1"
    & ControlChars.CrLf &
    "This is line 2")
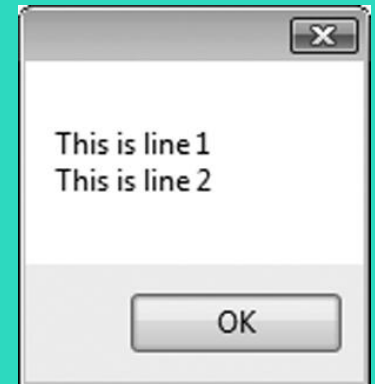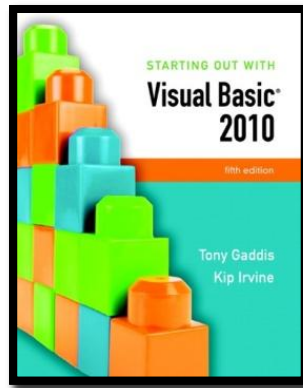
Section 4.9

THE SELECT CASE STATEMENT

In a **Select Case** statement, one of several possible actions is taken, depending on the value of an expression.

The **Select Case** Statement

- Similar to **If…Then…ElseIf**
  - Performs a series of tests
  - Conditionally executes the first true condition
- **Select Case** is different in that:
  - A single test expression may be evaluated
  - The test expression is listed once
  - The possible values of the expression are then listed with their conditional statements
- **Case Else** may be included and executed if none of the values match the expression

Select Case General Format

Select Case *TestExpression*
  [Case *ExpressionList*
    [one or more statements]]
  [Case *ExpressionList*
    [one or more statements]]
  ' Case statements may be repeated
  ' as many times as necessary.
  [Case Else
    [one or more statements]]
End Select

Select Case Statement Example

```
Select Case CInt(txtInput.Text)
        Case 1
                MessageBox.Show("Day 1 is Monday.")
        Case 2
                MessageBox.Show("Day 2 is Tuesday.")
        Case 3
                MessageBox.Show("Day 3 is Wednesday.")
        Case 4
                MessageBox.Show("Day 4 is Thursday.")
        Case 5
                MessageBox.Show("Day 5 is Friday.")
        Case 6
                MessageBox.Show("Day 6 is Saturday.")
        Case 7
                MessageBox.Show("Day 7 is Sunday.")
        Case Else
                MessageBox.Show("That value is invalid.")
End Select
```

Select Case Flowchart Example
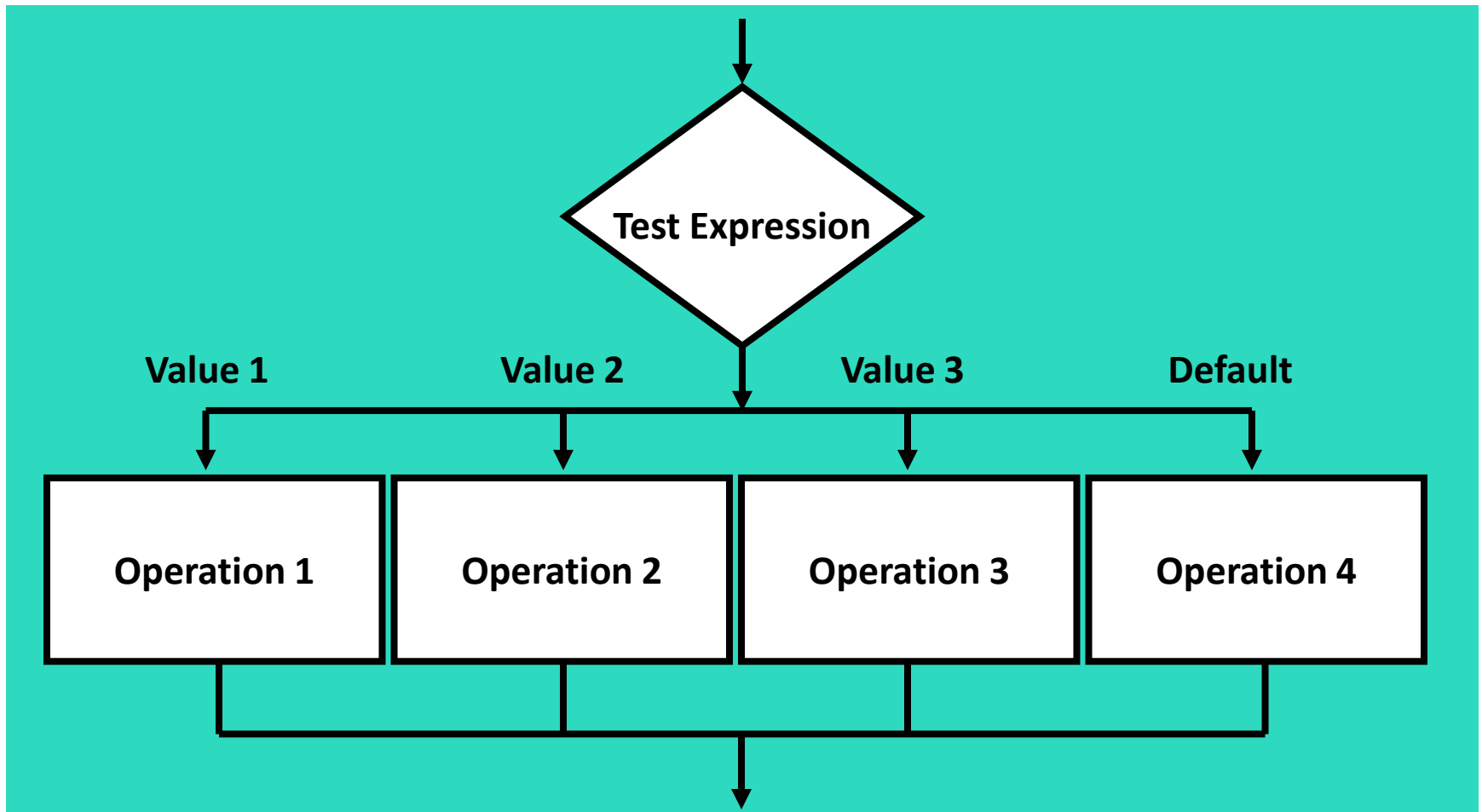
Select Case Pseudocode Example

```
Select Case Input
    Case 1
        Display Message "Day 1 is Monday."
    Case 2
        Display Message "Day 2 is Tuesday."
    Case 3
        Display Message "Day 3 is Wednesday."
    Case 4
        Display Message "Day 4 is Thursday."
    Case 5
        Display Message "Day 5 is Friday."
    Case 6
        Display Message "Day 6 is Saturday."
    Case 7
        Display Message "Day 7 is Sunday."
    Case Else
        Display Message "That value is invalid."
End Select
```

More about the Expression List: Multiple Expressions

- The **Case** statement's expression list can contain multiple expressions, separated by commas

```
Select Case intNumber
    Case 1, 3, 5, 7, 9
        strStatus = "Odd"
    Case 2, 4, 6, 8, 10
        strStatus = "Even"
    Case Else
        strStatus = "Out of Range"
End Select
```

More about the Expression List: String Values

- The **Case** statement can test string values

```
Select Case strAnimal
   Case "Dogs", "Cats"
      MessageBox.Show("House Pets")
   Case "Cows", "Pigs", "Goats"
      MessageBox.Show("Farm Animals")
   Case "Lions", "Tigers", "Bears"
      MessageBox.Show("Oh My!")
End Select
```

More about the Expression List: Relational Operators

- You can use relational operators in the **Case** statement
- The **Is** keyword represents the test expression in the relational comparison

```
Select Case dblTemperature
    Case Is <= 75
        blnTooCold = True
    Case Is >= 100
        blnTooHot = True
    Case Else
        blnJustRight = True
End Select
```

More about the Expression List: Ranges of Values

- You can determine whether the test expression falls within a range of values

- Requires the **To** keyword
  - Smaller number on the left
  - Larger number on the right
  - Numbers on each side are included in the range

```
Select Case intScore
   Case Is >= 90
      strGrade = "A"
   Case 80 To 89
      strGrade = "B"
   Case 70 To 79
      strGrade = "C"
   Case 60 To 69
      strGrade = "D"
   Case 0 To 59
      strGrade = "F"
   Case Else
      MessageBox.Show("Invalid Score")
End Select
```

Section 4.10

INTRODUCTION TO INPUT VALIDATION

Input validation is the process of inspecting input values and determining whether they are valid.

Validation Example

- Output is only as good as the input
    - "Garbage in, garbage out"
- Input validation is the process of inspecting user input to see that it meets certain rules
- The **TryParse** method verifies that an input value is in a valid numeric or date format
- Decision structures are often used to validate input

The **TryParse** Method

- Converts an input value to another format
  - Verifies that input of integers, decimals, dates, etc., are entered in an acceptable format
  - Returns Boolean value indicating **True** if conversion successful
  - Returns **False** if unsuccessful
- Each numeric variable type has a **TryParse** method
- Date & Boolean types include the **TryParse** method as well

Verify Integer Entry With **TryParse**

- Use **Integer.TryParse** method to convert value
  - **txtInput.Text** contains numeric string to convert
  - **intResult** receives converted value
  - **TryParse** returns **True** if input is an integer
  - **TryParse** returns **False** if input is not an integer

```
Dim intResult As Integer
If Integer.TryParse(txtInput.Text, intResult) Then
        lblMessage.Text = "Success!"
Else
        lblMessage.Text = "Error: an integer was not found"
End If
```

Checking Numeric Ranges

- Sometimes you need to check numeric input values to make sure they fall within a range

```
If intHours >= 0 And intHours <= 168 Then
    decGrosspay = intHours * decPayRate
Else
    MessageBox.Show("Invalid number of hours.")
End If


If intSpeed < 35 Or intSpeed > 60 Then
    MessageBox.Show("Speed violation!")
End If
```

Section 4.11

FOCUS ON GUI DESIGN: RADIO BUTTONS AND CHECK BOXES

Radio buttons appear in groups of two or more, allowing the user to select one of several options. A check box allows the user to select an item by checking a box, or deselect the item by unchecking the box.

Radio Buttons

- Used when only one of several possible options may be selected at one time
    - Car radio buttons select one station at a time
- May be placed in a group box
    - Group box defines a set of radio buttons
    - Can select only one button within a group box
    - Those on a form but not inside a group box are considered members of the same group
- Radio buttons have a Boolean **Checked** property and a **CheckChanged** event

Checking Radio Buttons in Code

```
If radCoffee.Checked = True Then
        MessageBox.Show("You selected Coffee")
ElseIf radTea.Checked = True Then
        MessageBox.Show("You selected Tea")
ElseIf radSoftDrink.Checked = True Then
        MessageBox.Show("You selected a Soft Drink")
End If
```
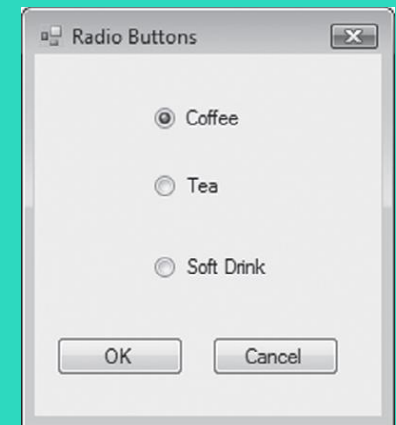
Check Boxes

- Unlike radio buttons, can select many check boxes at one time
- May also be placed in a group box
  - Not limited to one selection within a group box
  - Can select as many check boxes as you like within the same group box
- Check boxes also have a Boolean **Checked** property and a **CheckChanged** event
- Tutorial 4-9 provides radio button and check box examples

Checking Check Boxes in Code

```
' Determine which check boxes are checked.
If chkChoice4.Checked = True Then
    MessageBox.Show("You selected Choice 4.")
End If
If chkChoice5.Checked = True Then
    MessageBox.Show("You selected Choice 5.")
End If
If chkChoice6.Checked = True Then
    MessageBox.Show("You selected Choice 6.")
```
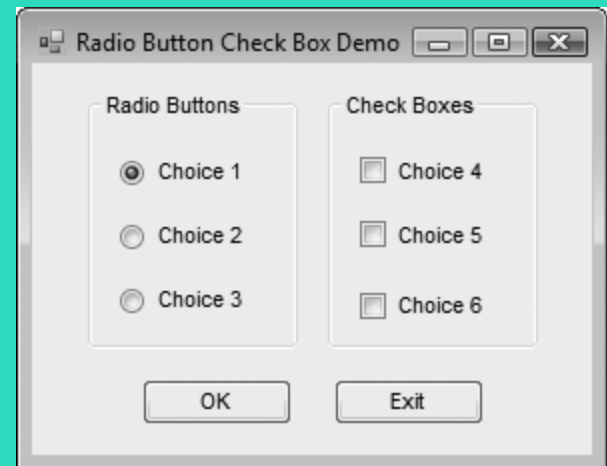
Section 4.12

FOCUS ON PROGRAM DESIGN AND PROBLEM SOLVING: BUILDING THE *HEALTH CLUB MEMBERSHIP FEE CALCULATOR APPLICATION*

In this section you build the Health Club Membership Fee Calculator application. It will use features discussed in this chapter, including decision structures, radio buttons, and check boxes.

Health Club Fee Calculator Form

Calculate Button Click Event Flowchart
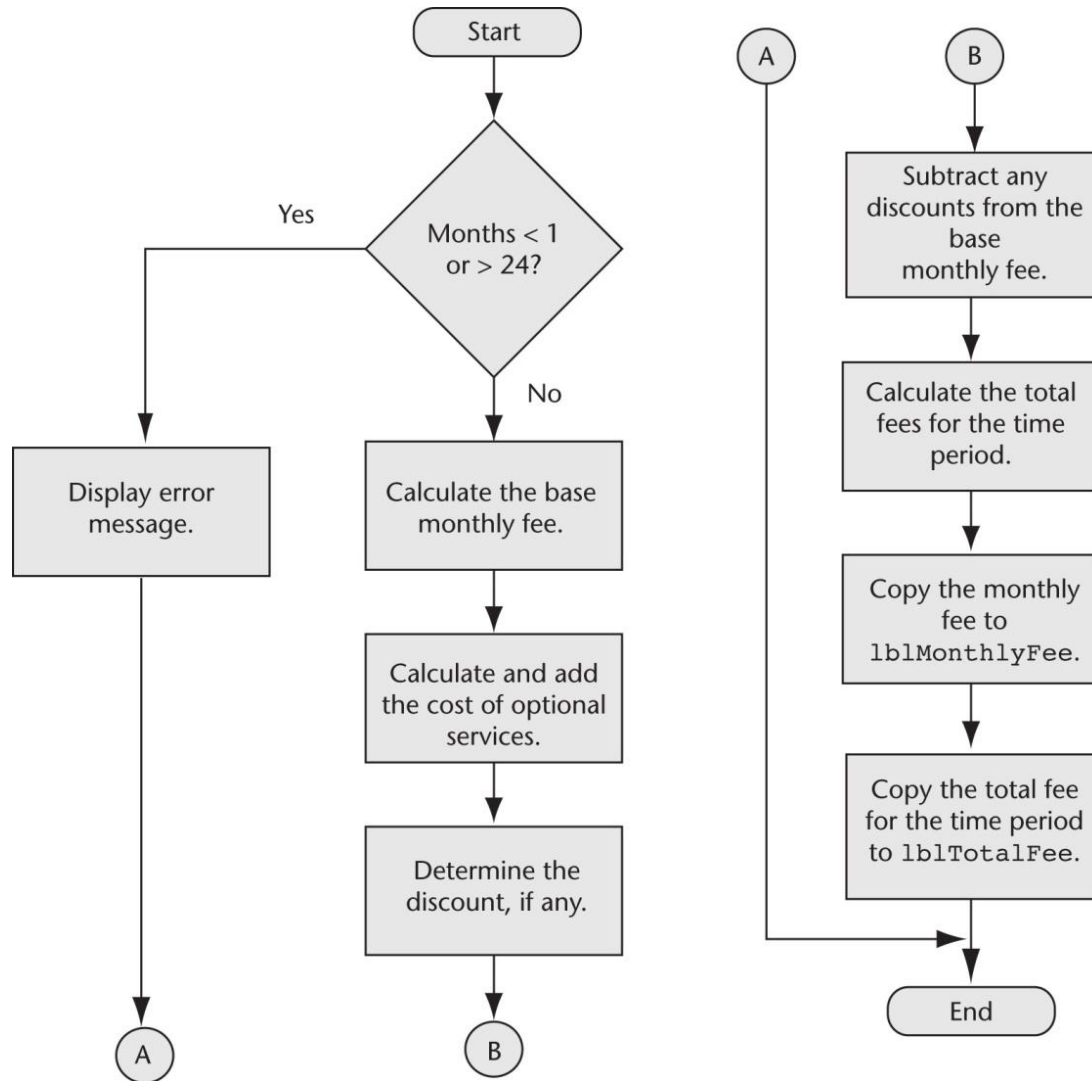
Base Monthly Fee Calculation Flowchart & Pseudocode

If Member is an Adult Then
Monthly Base Fee = 40
ElseIf Member is a Child Then
Montlhy Base Fee = 20
ElseIf Member is a Student Then
Monthly Base Fee = 25
ElseIf Member is a Senior Citizen Then
Monthly Base Fee = 30
End If
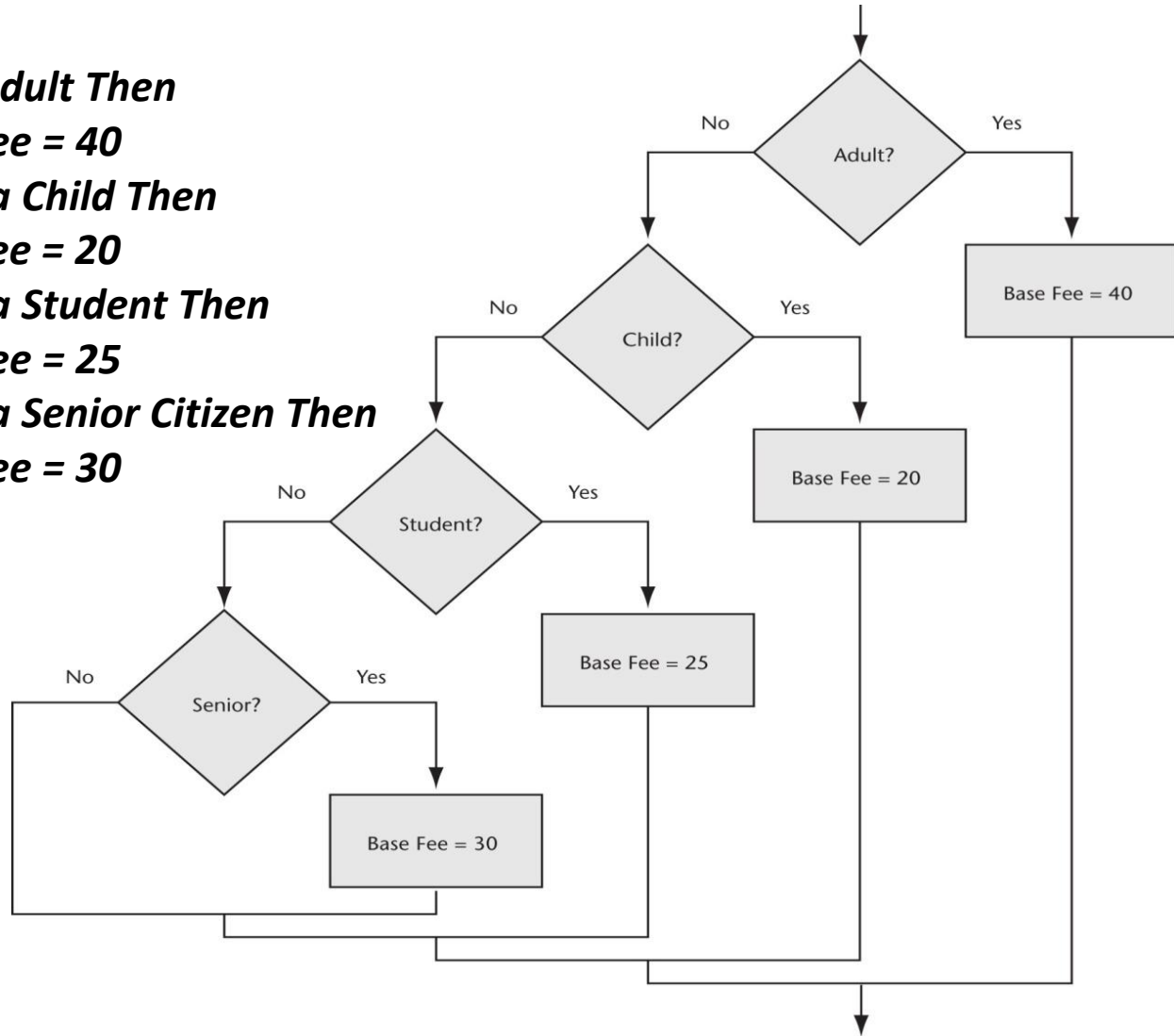
Calculate Optional Services Flowchart & Pseudocode

If Yoga is selected Then
* Add 10 to the monthly base fee*
End If
If Karate is selected Then
* Add 30 to the monthly base fee*
End If
If Personal Trainer is selected Then
* Add 50 to the monthly base fee*
End If
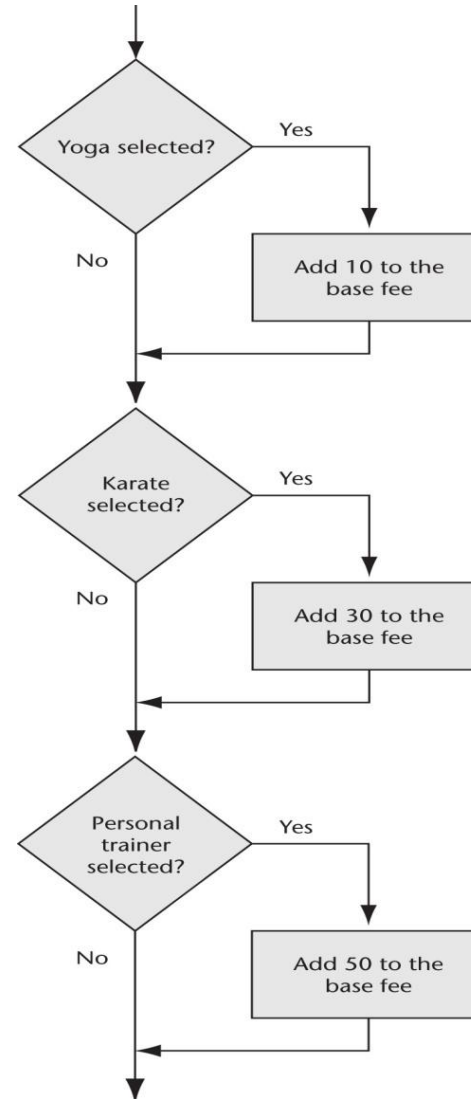
The Completed Membership Fee Calculator Form

Test Data for the Membership Fee Calculator

Type of Membership	Monthly Fee	Total
Standard adult with yoga, karate, and personal trainer for 6 months	$130.00	$780.00
Child with karate for 3 months	$50.00	$150.00
Student with yoga for 12 months	$35.00	$420.00
Senior citizen with karate and personal trainer for 8 months	$110.00	$880.00