Chapter 4
Decisions and Conditions

Chapter Objectives

- Use if...else statements to control the flow of logic
- Understand and use nested if statements
- Evaluate Boolean expressions using the relational or comparison operators
- Combine expressions using logical operators && (and), || (or), and ! (not)

Chapter Objectives – cont’d

- Use a switch...case structure for multiple decisions
- Call an event handler from other methods
- Create message boxes with multiple buttons and choose alternate actions based on the user response
- Debug projects using breakpoints, stepping program execution, and displaying intermediate results
### if Statement – General Form

- Condition to test is placed in parentheses
- if and else statements do not have semicolons
  - A semicolon terminates the statement
  - Any statement following the semicolon executes unconditionally
- Indent for readability and clarity

```csharp
if (condition)
{
    // Statement(s)
} [else
{
    // Statement(s)
}]
```

### if Statements

- A decision made by the computer formed as a question
- If the condition is true
  - Do one thing
- If the condition is false
  - Do something else

```csharp
if the sun is shining (condition)
go to the beach (action to take if condition is true)
else
go to class (action to take if condition is false)
```

### if Statements – cont’d

- In an if statement
  - When the condition is true
    - Only the statement following the if is executed
  - When the condition is false
    - Only the statement following the else clause, if present, is executed
- Use braces to include multiple statements in the if or else portion of statement
- Always use braces**
  - Good programming practice
if Statement – Examples

- Type an `if` statement
  - Press Enter
  - Editor places insertion point on blank line, indented
- Multiple statements must be in braces
  - Editor places braces directly under `if` or `else` statements

```csharp
unitsDecimal = decimal.Parse(unitsTextBox.Text);
if (unitsDecimal < 32m)
{
    freshmanRadioButton.Checked = true;
}
else
{
    freshmanRadioButton.Checked = false;
}
```

Charting if Statements (FYI)

- A Uniform Modeling Language (UML) activity diagram is a useful tool for showing the logic of an `if` statement
- Helps to organize thoughts and design projects more quickly
- UML includes several types of diagrams
  - Activity diagram is a visual planning tool for decisions and actions for an entire application or a single method

Boolean Expressions

- The test in an `if` statement is based on a `Boolean expression (condition)` using relational operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Relation tested</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
<td>decimal.Parse(amountTextBox.Text) &gt; limitDecimal</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
<td>int.Parse(salesTextBox.Text) &lt; 10000</td>
</tr>
<tr>
<td><code>==</code></td>
<td>equal to (note: this is the true “equal sign”</td>
<td>passwordTextBox.Text == &quot;101&quot;</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal to</td>
<td>freshmanRadioButton.Checked != true</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
<td>int.Parse(quantityTextBox.Text) &gt;= 500</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
<td>countInteger &lt;= maximumInteger</td>
</tr>
</tbody>
</table>
Testing for True or False

- Shortcuts when testing for true or false
- C# evaluates expression in if statement
- If condition is a Boolean variable or property, it holds true or false values

if (blueRadioButton.Checked == true)
    is equivalent to
if (blueRadioButton.Checked)

Comparing Uppercase and Lowercase Characters

- String comparisons are case sensitive
  - Uppercase "Y" is not equal to lowercase "y"
- User input may be uppercase, lowercase, or a combination
- To compare uppercase and lowercase characters
  - Convert user input to all uppercase or all lowercase
    - Use ToUpper and ToLower methods of the string class to convert
- Compare to the correct case of the literal
  - if (nameTextBox.Text.ToUpper() == "PROGRAMMING")
    - // Do something.

Compound Boolean Expressions

- Test more than one condition by joining conditions with logical operators

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(or)</td>
</tr>
<tr>
<td></td>
<td>int.Parse(numberLabel.Text) == 1</td>
<td></td>
</tr>
<tr>
<td>&amp;&amp; (and)</td>
<td>Both expressions must be true for the entire expression to be true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>int.Parse(numberTextBox.Text) &gt; 0 &amp;&amp; int.Parse(numberTextBox.Text) &lt; 10</td>
<td></td>
</tr>
<tr>
<td>! (not)</td>
<td>Reverses the Boolean expression so that a true expression will evaluate false and vice versa</td>
<td></td>
</tr>
<tr>
<td></td>
<td>! foundBool</td>
<td></td>
</tr>
</tbody>
</table>
Combining Logical Operators

- When combining logical operators
  - Can use multiple && and || operators
  - If both && and || are used, && is evaluated before ||
  - Change the order of evaluation with parentheses

```csharp
if (saleDecimal > 1000.0m || discountRadioButton.Checked && stateTextBox.Text.ToUpper() != "CA")
{
    //Code here to calculate the discount.
}
```

Nested if Statements

- An if statement that contains additional if statements
- Nest if in the true block
- Nest if in the else block
  - Code is simpler when else if is used
- if statements can be nested as deeply as desired
  - Projects are difficult to follow and may not perform correctly when ifs are too deeply nested

```csharp
if (tempInteger <= 32)
{
    commentLabel.Text = "Freezing";
}
else if (tempInteger > 80)
{
    commentLabel.Text = "Hot";
}
else
{
    commentLabel.Text = "Moderate";
}
```

Coding an else if

- To nest in the else statement, use else if (on one line), or as two separate statements

```csharp
if (tempInteger <= 32)
{
    commentLabel.Text = "Freezing";
}
else if (tempInteger > 80)
{
    commentLabel.Text = "Hot";
}
else
{
    commentLabel.Text = "Moderate";
}
```

Note that the indentation changes when else if is on one line.
Checking the State of a Radio Button Group

To check the state of multiple radio buttons, use a nested if in a button’s click event handler

```csharp
if (freshmanRadioButton.Checked)
    freshmanCountInteger++;
else if (sophomoreRadioButton.Checked)
    sophomoreCountInteger++;
else if (juniorRadioButton.Checked)
    juniorCountInteger++;
else if (seniorRadioButton.Checked)
    seniorCountInteger;
```

Only one radio button can be selected at a time.

Enhancing Message Boxes

- Control the format of the message
- Display multiple buttons
  - Check to see which button the user clicked
  - Perform alternate actions depending on the user’s selection

Displaying the Message String

- Message string displayed in a message box can be
  - A string literal (enclosed in quotes)
  - A string variable
- Create a variable for the message
- Format the message
- Call the `Show` method
  - Above three steps make code easier to read and follow
Combining Values into a Message String

- Concatenate a literal and the value from a variable
  - Include an extra space inside the literal to separate the variable's value from the literal

```csharp
string messageString = "Total Sales: " + totalDecimalSales.ToString("C");
MessageBox.Show(messageString, "Sales Summary", MessageBoxButtons.OK);
```

Creating Multiple Lines of Output

- Long messages wrap to a second line
- Insert a `NewLine` (`\n`) character in the string message
  - `\n` is known as an escape sequence.
    - See next slide

```csharp
string formattedTotalString = totalSalesDecimal.ToString("N");
string formattedAvgString = averageSaleDecimal.ToString("N");
string messageString = "Total Sales: " + formattedTotalString + "\n" + "Average Sale: " + formattedAvgString;
MessageBox.Show(messageString, "Sales Summary", MessageBoxButtons.OK);
```

Other Character Escape Sequences

- Other constants are available from the character escape sequence `\` list

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>Includes a single quote in a character literal</td>
</tr>
<tr>
<td>&quot;</td>
<td>Includes a double quote in a string literal</td>
</tr>
<tr>
<td>\</td>
<td>Includes a backslash in a string literal</td>
</tr>
<tr>
<td>\n</td>
<td>New line</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace character</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal tab</td>
</tr>
</tbody>
</table>
Choose buttons to display in a message box using the MessageBoxButtons constants
- Returns a DialogResult object

Capture information about the Show method's outcome
- Declare a variable to hold an instance of the DialogResult type
  DialogResult whichButtonDownResult;
- Assign the return value of the Show method to the new variable

if (whichButtonDownResult == DialogResult.Yes)
{
    // Code to clear the order.
}
### Input Validation

- Validation – Checking to verify appropriate values have been input
- Validation may verify:
  - Input is numeric
  - A specific value
  - A range of values
  - Required items are entered

### Checking for a Range of Values

- Check reasonableness of a value
  - Example – A company does not allow more than 10 hours to be worked in a single day
  ```csharp
  if (int.Parse(hoursTextBox.Text) > 10)
  {
      MessageBox.Show("Too many hours.", "Invalid Data",
                      MessageBoxButtons.OK);
  }
  ```

### The switch Statement

- Provides flexible and powerful alternative to if statements for testing multiple conditions
  ```csharp
  switch(expression)
  {
  case testValue1:
    statements(s);
    break;
  [ case testValue2:
    statements(s);
    break;]
  [default:]
    statements(s);
    break;
  }
  ```
The switch Statement – cont’d

- The test values must be the same data type as the expression
- There is no limit to the number of case blocks and no limit to the number of statements following each case statement
- default clause is optional, code executes only if no other case is matched
- The break statements for each case, including the default, are required
- Only the statements in the first matched case execute

Calling Event Handlers

- Never duplicate code
- Write code once in an event-handling method
  - "Call" the method from another method by naming the method
  - Entire method is executed
  - Execution returns to statement following the call
  - Include parentheses, leave empty if no arguments
  - If method requires arguments, place arguments within parentheses

Debugging C# Projects

- Quickly check the current value of a variable, a control, a Boolean expression, or an arithmetic expression
  - Break execution using a breakpoint
  - Point to the name of the expression
  - Current contents of expression pop up in a small label called a DataTip, when the expression is in scope
Stepping through Code

- The best way to debug a project is to trace program execution line by line
  - Break execution with a breakpoint or choose a stepping command
    - Step Into (F11) - Executes next line of code, continue stepping through code by repeatedly choosing Step Into command
    - Step Over (F10) - Executes next line of code, displays only the lines of code in the current method being analyzed
    - Step Out (Shift + F11) - Rapidly executes a called method, returns to debug mode at the statement following the call