Chapter 4
Advanced Programming with Visual C#

1. **Debugging C# Applications**
   - Design-time errors
   - Run-Time errors
   - Logical errors
2. **Creating Controls On-the-Fly**
3. **Deploying C#**

Debugging C# Applications

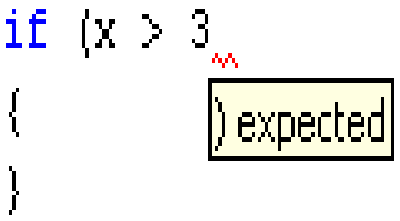
- **Debugging** :
  - The process of trying to track down errors in your programs.
  - Handling potential errors that may occur in your programs.
  - There are three types of errors.
    - Design-time error
    - Runtime error
    - Logical error

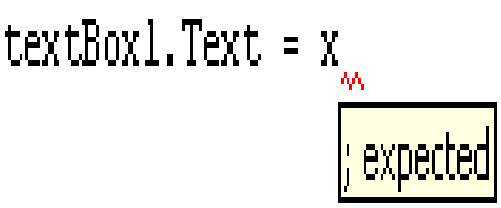
Debugging C# Applications (Cont'd)

- **Design-time errors**
  - Errors that don't allow your program to run.
  - In such case, you will get popup message telling that ***"there were build errors, would you like to continue".***
  - Design-Time errors are easy enough to spot
    - ✓ because the VS software will underline them with a ***wavy colored*** line
  - There are three different Waye colored lines
    - ✓ Red
    - ✓ Blue
    - ✓ Green

Debugging C# Applications (Cont'd)

- ***Red wavy* lines**
  - Syntax errors
  - such as a missing semicolon at the end of a line, or a missing curly bracket in an IF Statement.
  - Example





Debugging C# Applications (Cont'd)

- ***Blue wavy* lines**
- are known as **Edit and Continue** issues, meaning that you can make change to your code without having to stop the program altogether.

```
textBox2.Text = x;
```

The name 'textBox2' does not exist in the current context

Debugging C# Applications (Cont'd)

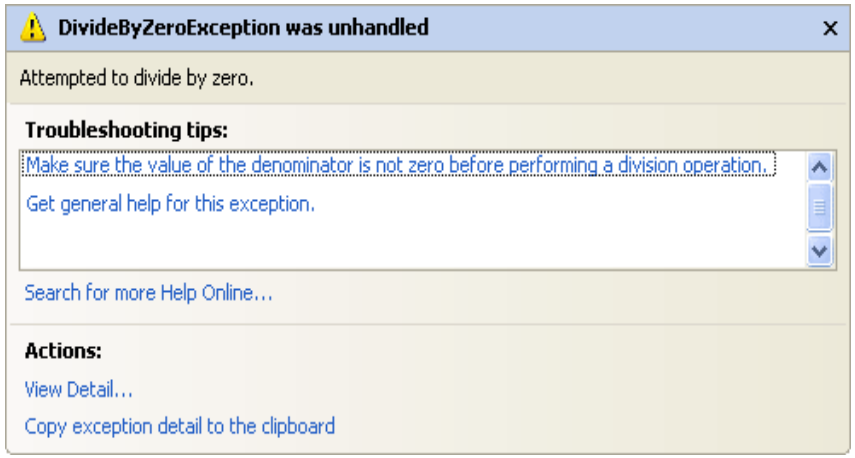
- ***Green wavy* lines**
  - are **Compiler Warnings**.
  - when C# spots something that could potentially cause a problem, such as declaring a variable that's never used.

```
private void button1_Click(object sender, EventArgs e)
{

    string CompilerWarning = "";
           The variable 'CompilerWarning' is assigned but its value is never used
    textBox1.Text = "";

}
```

Debugging C# Applications (Cont'd)

- **Run Time Errors**
  - The program starts up OK, but crashes at some point.
  - Example

```
private void button1_Click(object sender, EventArgs e)
{

    int Num1 = 10;
    int Num2 = 0;
    int answer;

    answer = Num1 / Num2;

}
```



Debugging C# Applications (Cont'd)

- **Logic Errors**
  - Error made in the programming logic
  - Errors where you did not get the result you were expecting
  - The program runs into completion successfully with wrong result.

```
private void button1_Click(object sender, EventArgs e)
{

    int startLoop = 11;
    int endLoop = 1;
    int answer = 0;

    for (int i = startLoop; i < endLoop; i++)
    {
        answer = answer + i;
    }

    MessageBox.Show("answer =" + answer.ToString());
}
```

- Debugging tools that help you track down logic errors
  - Breakpoints
  - Locals Window
  - Example

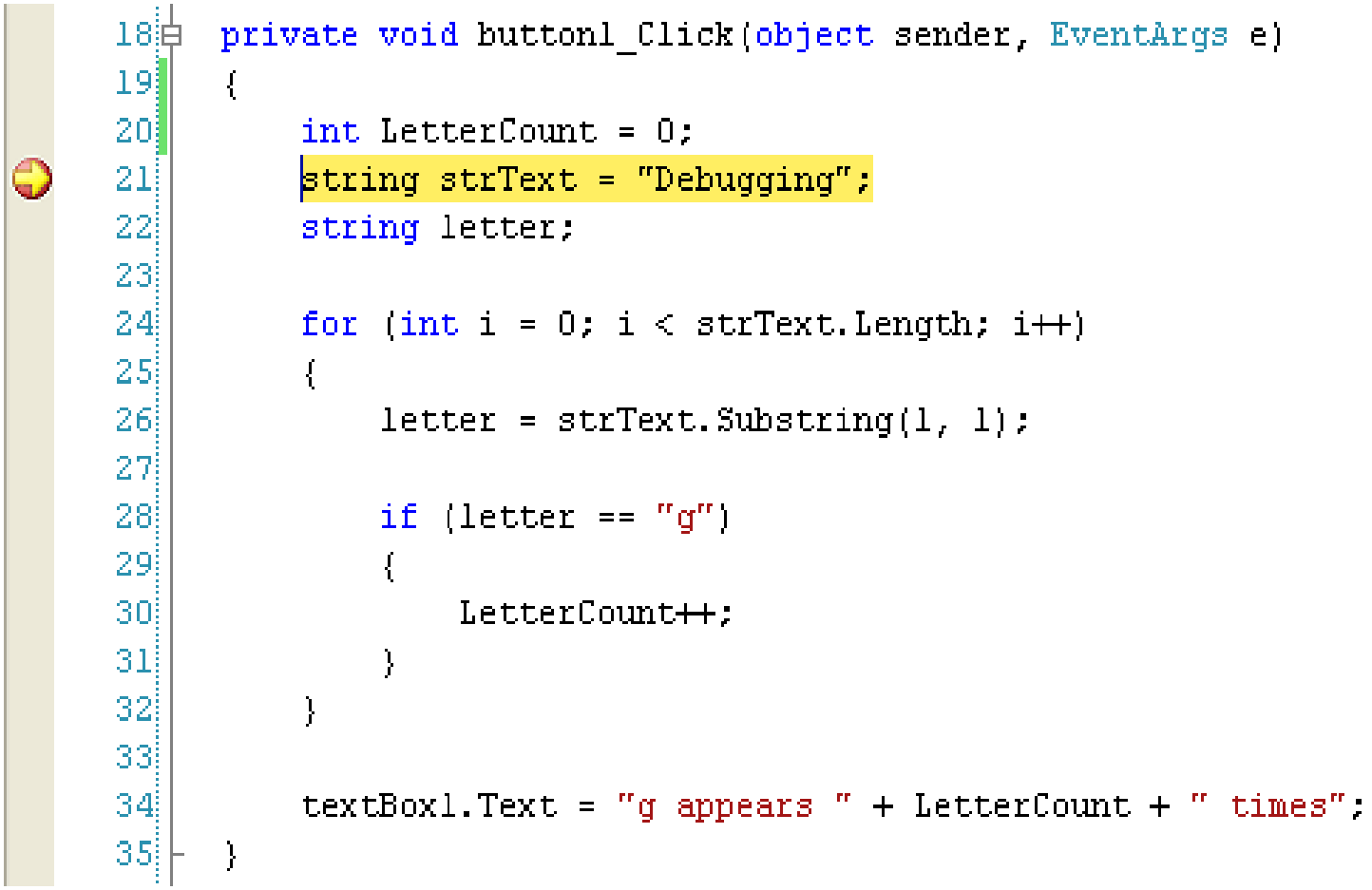
Logical errors can be very difficult to track down!

```
private void button1_Click(object sender, EventArgs e)
{
    int LetterCount = 0;
    string strText = "Debugging";
    string letter;

    for (int i = 0; i < strText.Length; i++)
    {
        letter = strText.Substring(1, 1);

        if (letter == "g")
        {
            LetterCount++;
        }
    }

    textBox1.Text = "g appears " + LetterCount + " times";
}
```

Debugging C# Applications (Cont'd

- **Breakpoints**
  - This is where you tell C# to halt your code, so that you can examine what is in your variables.
  - To add a Breakpoint:
    - click in the margins to the left of a line of code.
  - To see what a breakpoint does:
    - run your program and then click your button
    - C# will display your code as follows:

Debugging C# Applications (Cont'd)



```
18  private void button1_Click(object sender, EventArgs e)
19  {
20      int LetterCount = 0;
21      string strText = "Debugging";
22      string letter;
23
24      for (int i = 0; i < strText.Length; i++)
25      {
26          letter = strText.Substring(1, 1);
27
28          if (letter == "g")
29          {
30              LetterCount++;
31          }
32      }
33
34      textBox1.Text = "g appears " + LetterCount + " times";
35  }
```

Debugging C# Applications (Cont'd)

- **Breakpoints**
  - Press F10 on your keyboard and the yellow arrow will jump down one line. Keep pressing F10 and trace values of each variable in the program to track where the error has happened.

Debugging C# Applications (Cont'd)

- **The Locals Window**
  - The Locals Window keeps track of what is in local variables
    - Add a new breakpoint
    - Run your program
    - click the button
    - When you see the yellow highlighted line, click the **Debug** menu at the top of C#.
    - click **Windows > Locals**
    - You should see the following window appear at the bottom of your screen:

Debugging C# Applications (Cont'd)

- The Locals Window

```
for (int i = 0; i < strText.Length; i++)
{
     letter = strText.Substring(0, 1);

     if (letter == "g")
     {
          LetterCount++;
     }
}
```

Locals

Name	Value
this	{debugging.Form1, Text: Form1}
sender	{Text = "button1"}
e	{X = 33 Y = 13 Button = Left}
i	0
LetterCount	0
strText	"Debugging"
letter	"D"

Call Stack | Immediate Window | Error List | Locals | Watch

Debugging C# Applications (Cont'd)

- Keep pressing F10 and the values will change
- Here's what is inside of the variables after a few spins round the loop:

```
for (int i = 0; i < strText.Length; i++)
{
    letter = strText.Substring(0, 1);

    if (letter == "g")
    {
        LetterCount++;
    }
}
```

Locals

Name	Value
this	{debugging.Form1, Text: Form1}
sender	{Text = "button1"}
e	{X = 33 Y = 13 Button = Left}
i	3
LetterCount	0
strText	"Debugging"
letter	"D"

Call Stack | Immediate Window | Error List | Locals | Watch

Creating Controls On-the-Fly

- **Dynamically Adding Controls**
  - how can you add controls during runtime where the tools from Visual Studio are not accessible?
    - Ans. You can do so by adding them programmatically
  - To demonstrate create an application that adds control to the form based on numbers typed by the user
  - The program will dynamically add labels and text boxes to the form.
  - Create a new Windows Forms Application. Add a label, text box and a button as seen below.

Deploying C# Windows Applications in Debug Mode

- **Steps to Create Setup and Deployment Project in .Net with VS 2010**
  - **Step 1**
    - Create your own windows application, complete the design and coding and build the solution of the project in release mode.
  - **Step 2**
    - Check the Release/debug folder for the file "ProjectName.exe". Here in this example we have the project name as sample so we can find a file with the name Sample.exe. Double click the executable file and check the example.

Deploying C# Windows Applications in Debug Mode (Cont'd)

- **Step 3**
  - Now let's create an installer for the same windows application. Right click on the solution and add a new project to your solution like in following figure:
  - Select the "Other Project Types" -> "Setup and Deployment" -> "Setup project". Here we have the setup project for example as "SampleSetup".
- **Step 4**
- Right click on Application Folder and add the ***Sample.exe*** project application file inside the "Application Folder".

Deploying C# Windows Applications in Debug Mode (Cont'd)

- **Step 5**
- Now build the SampleSetup project mode.
- **Step 6**
- The setup file created in C:\Users\ Administrator\Desktop\Sample\ SampleSetup\Debug folder of the project specified path.
- **Step 7**

  Run the setup and install it.

Deploying C# Windows Applications in Release Mode

- Right click on solution name -> Add -> New Project -> Installed Templates -> Other Projects Template -> Setup and Deployment -> Visual Studio Installer -> Setup Project -> Give Setup Name -> Click Ok.

Deploying C# Windows Applications in Release Mode (Cont'd)

- Go to Setup Project -> Right Click -> Add -> Project Output -> Project -> Name of Your Project -> Select Primary Output -> Click Ok.
- Go to & Click Build Menu -> Click Batch Build -> Check the Project Configuration to build -> Click Build.
- Now, Set up is Deployed.


End of theory class!!
06/04/2018 E. C