



## CST-150 Activity 3 Guide

### Contents

Part 1 .....	2
Read a Text File, Process the Contents, and Write the Results Back to File .....	2
Overview .....	2
Execution .....	3
1. Start a new Visual Studio Project. ....	3
2. Verify the application is working correctly. ....	6
3. Configure the Form Layout (FrontEnd Design). ....	6
4. Configure the Open File Dialog .....	9
5. Configure the Event Handler .....	10
6. Start coding the backend. ....	11
7. Configure the Text File .....	13
8. Continue coding the backend. ....	13
9. Format the label. ....	15
10. Add in the Header Text. ....	17
11. Submit the Activity as described in the digital classroom. ....	18

## Part 1

### Read a Text File, Process the Contents, and Write the Results Back to File

#### Overview

Write a Windows Forms Application that uses a file dialog control to allow the user to select a multi-line text file for input, processing the contents of the file and displaying the results in a label. The instructor will provide this file that must be used for this activity. The file is a multi-line comma delimited file with text and integer values. The objective is to read from the file one line at a time using appropriate looping structure extracting text and decimal values.

The program that is designed will look similar to Figure 1.

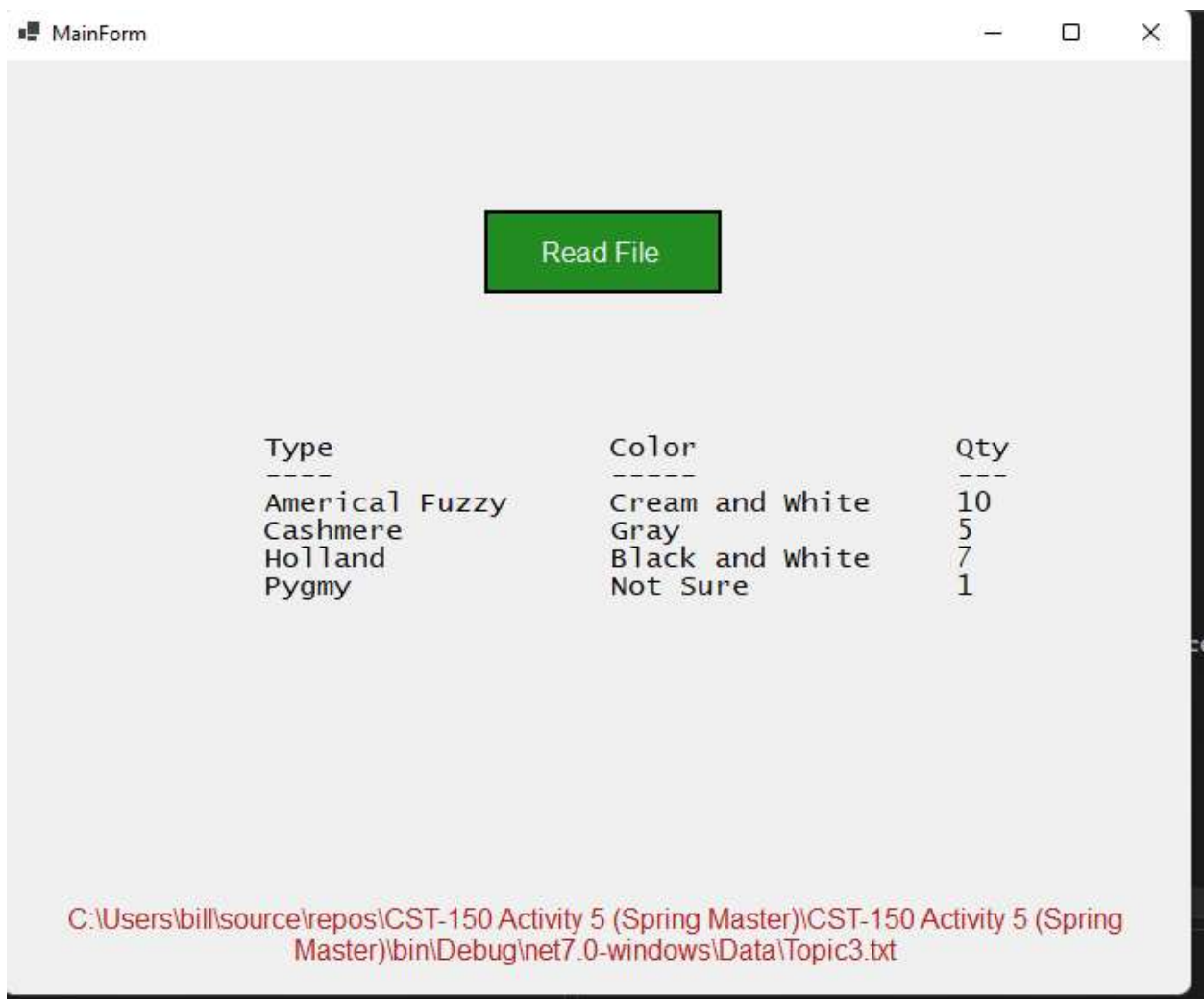


Figure 1: Final result of this activity.

## Execution

Execute this assignment according to the following guidelines:

1. Start a new Visual Studio Project.
  - a. Draw a flowchart to visually describe the flow of the program. The flowchart is the blueprint to design the program therefore, flowcharting is the first step in designing any program. The flowchart must be created using a flowchart design software as directed by your instructor. Take a screenshot of the flowchart and provide it as one of the deliverables.
  - b. Start Visual Studio and select "Create a new project" as shown in Figure 2.

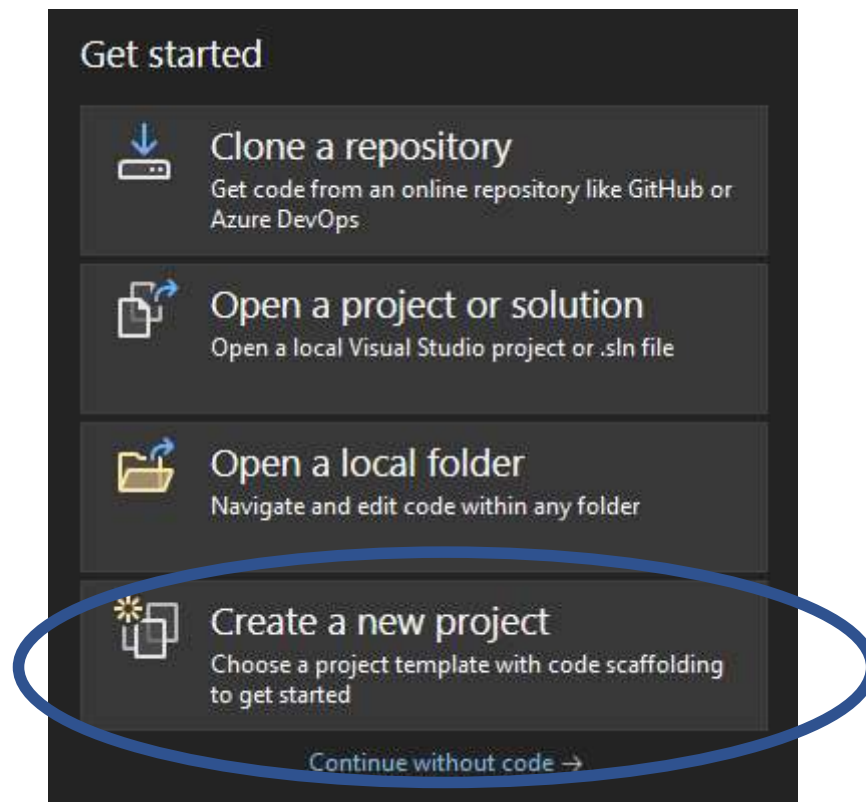


Figure 2: Select "Create a new project."

- c. Select the Project Template "WindowsFormsApp" as shown in Figure 3. Make sure to select "Windows Forms App." DO NOT select the one with (.NET Framework).

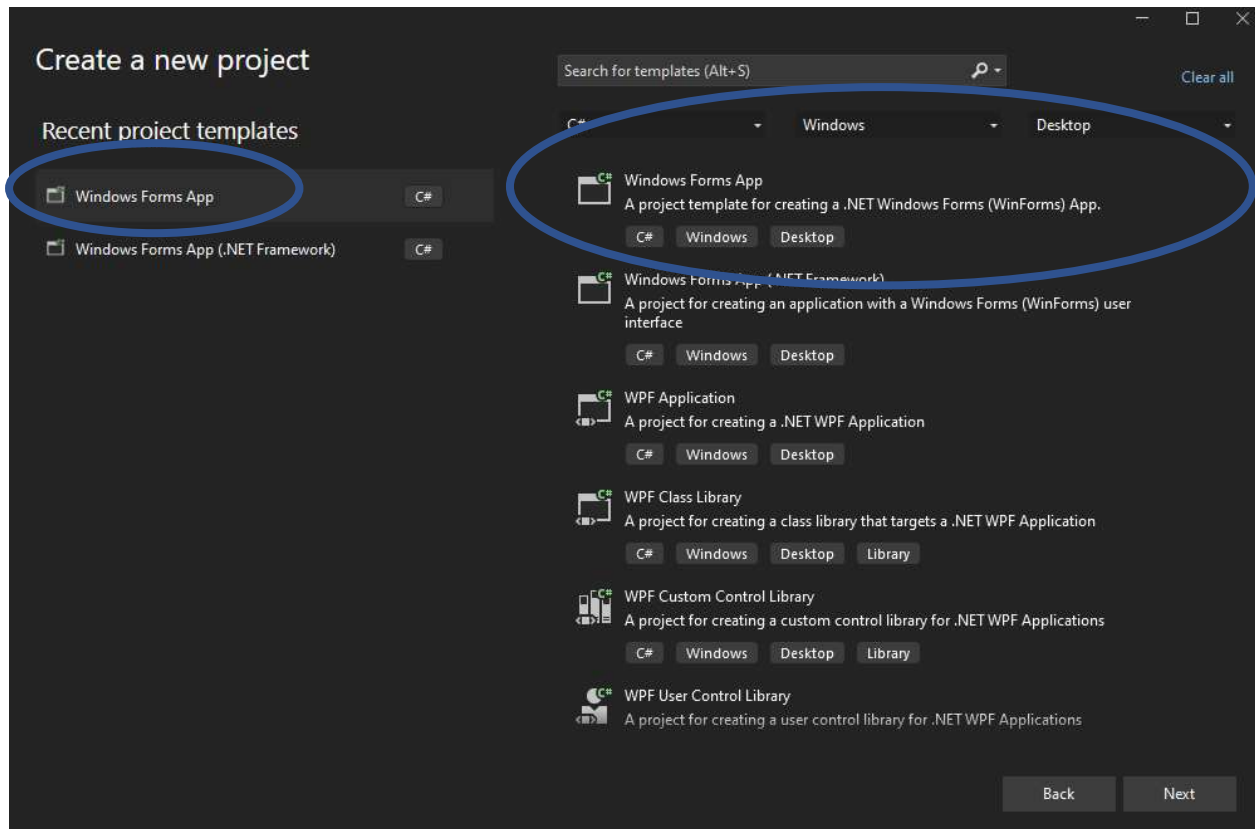


Figure 3: Select the Project Template.

- d. Enter the Project name: "CST-150 Activity 3" as shown in Figure 4 followed by the "Next" button. Make sure the check box is not checked for "Place solution and project in the same directory."

Configure your new project

Windows Forms App C# Windows Desktop

Project name

CST-150 Activity 3

Location

C:\Users\bill\source\repos\Activities

Solution name ⓘ

CST-150 Activity 3

☐ Place solution and project in the same directory

*Figure 4: Enter Project Name.*

- e. Select the Framework as is shown in Figure 5 followed by the "Create" button. These in class applications have all been tested using .NET 7.0. (The instructor may have the class use a different .NET version.)

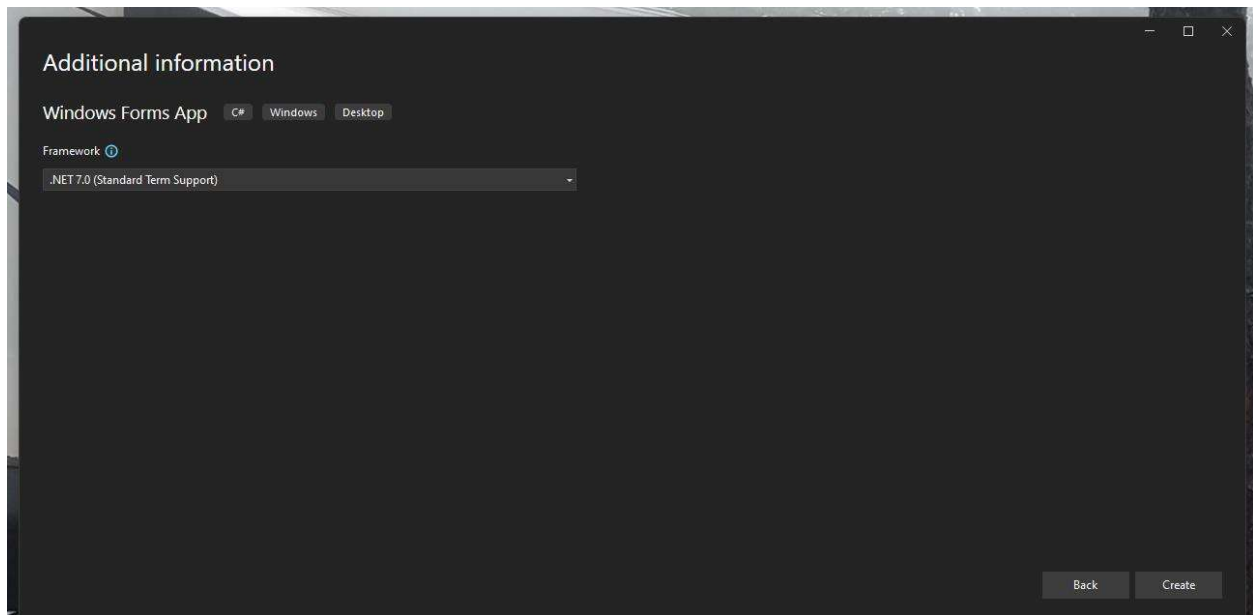


Figure 5: Select Framework.

2. Verify the application is working correctly.
  - a. Follow the steps outlined in Activity 3.
3. Configure the Form Layout (FrontEnd Design).
  - a. Rename the form name to "FrmMain.cs."
  - b. Change the form Text property to "MainForm" as shown in Figure 6 and Figure 7.

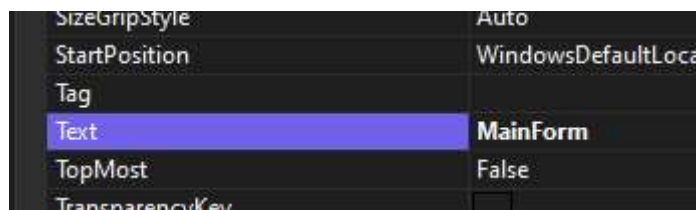


Figure 6: Configure the Form Text Property.

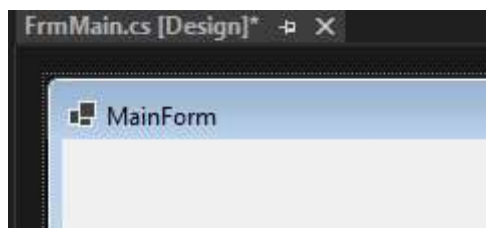


Figure 7: Form Text Property

- c. Place a button control on the form and change the text property to "Read File" as shown in Figure 8.
- d. Configure the button properties as follows and shown in Figure 9:
  - BackColor: ForestGreen (Web Color)
  - ForeColor: WhiteSmoke (Web Color)
  - Size: W-137 and H-48
  - Cursor: Hand
  - FlatAppearance:
  - Mouse over: LimeGreen
  - BorderColor: Black
  - BorderSize: 1
  - MouseDown: DarkOliveGreen
  - To activate FlatAppearance change FlatStyle to "Flat"

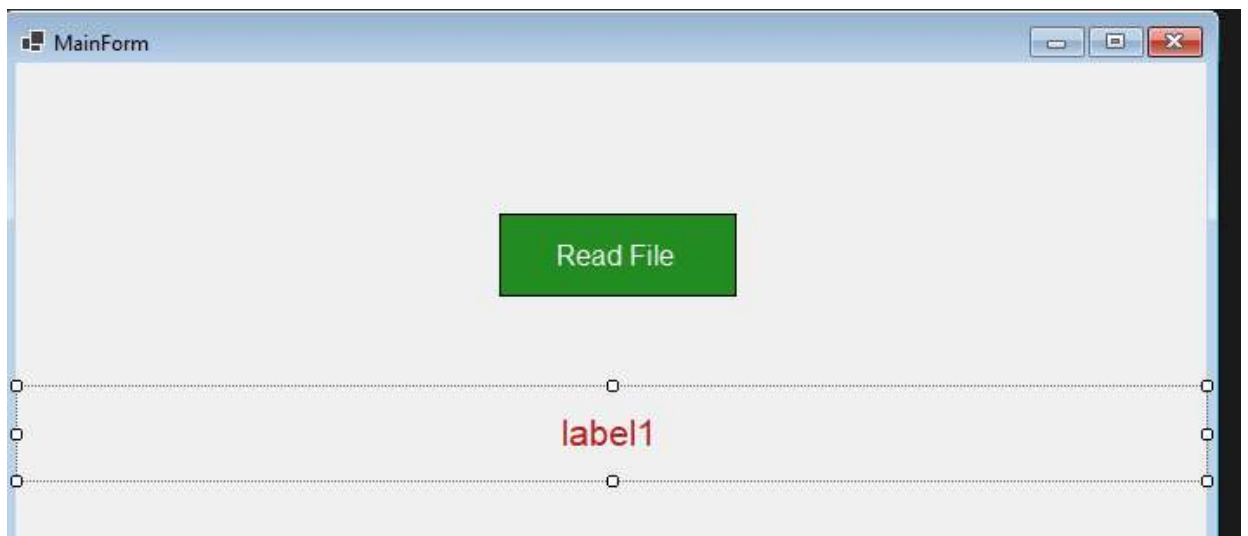


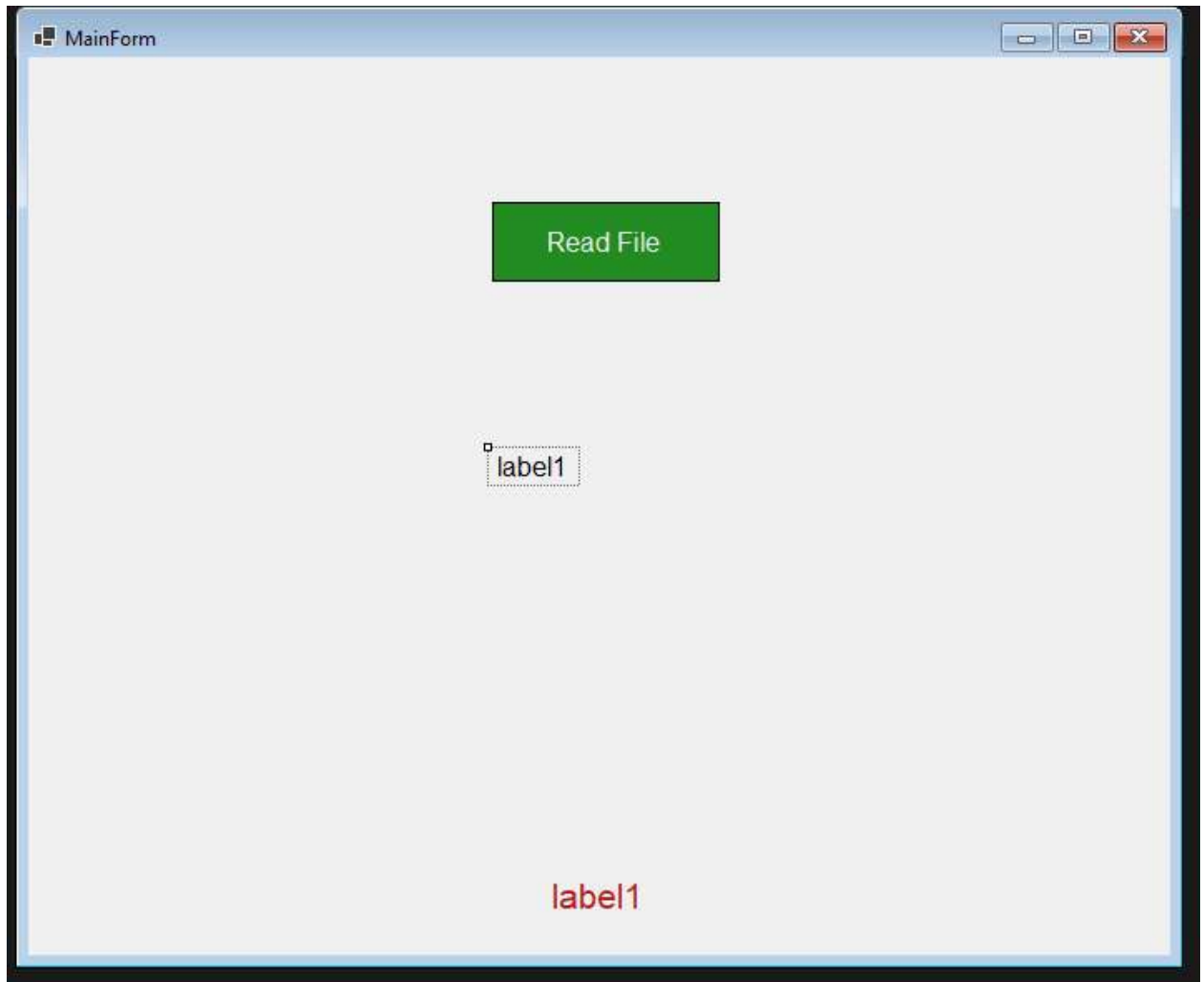
Figure 8: Main Form

Dock	None
Enabled	True
FlatAppearance	
BorderColor	Black
BorderSize	1
MouseDownBackColor	DarkOliveGreen
MouseOverBackColor	LimeGreen
FlatStyle	Flat
Font	Arial, 12pt
ForeColor	WhiteSmoke
GenerateMember	True
Image	(none)

Figure 9: Button Properties

8. Place a label control at the bottom of the form as shown in Figure 10.
9. Configure the label properties as follows:
  - Name = "lblSelectedFile"
  - 12pt
  - Forecolor = Firebrick from web colors
  - Autosize to false
  - TextAlign to MiddleCenter
  - Stretch the label the full width.
9. Place a second label control in the middle of the form as shown in Figure 10.
1. Configure the label properties name to "lblResults."





*Figure 10: Label placement on form.*

4. Configure the Open File Dialog
  - a. Using Figure 11, follow the instructions to place an OpenFileDialog into the application.
  - b. Open File Dialog provides the popup screen, so the user can select the text file to open.

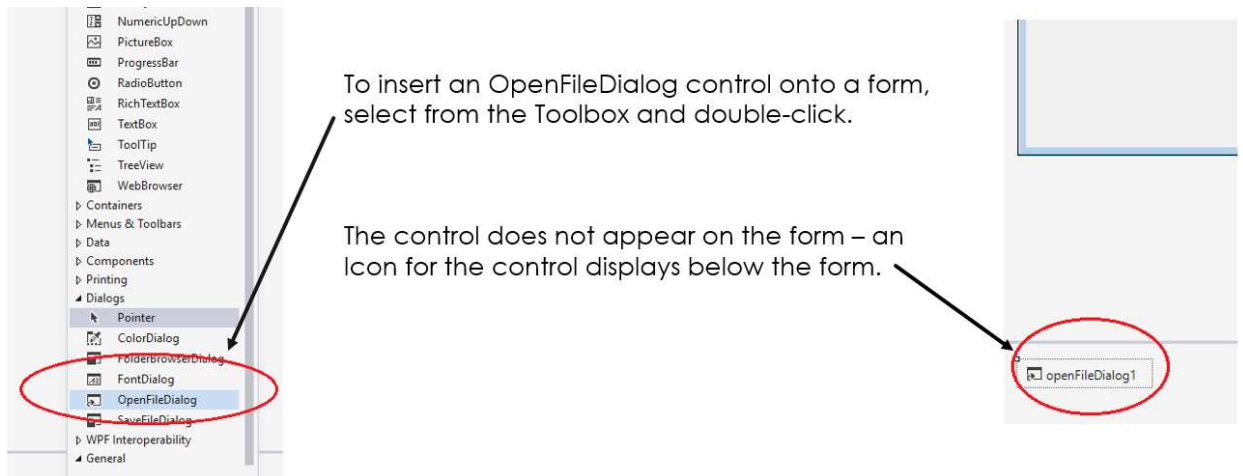


Figure 11: OpenFileDialog

- c. Initialize the File Dialog in the constructor as shown in Figure 12.

```

/// <summary>
/// Class Constructor
/// This method is the first method to get invoked.
/// </summary>
1 reference
public FrmMain()
{
    InitializeComponent();
    // Set the properties for the selectFileDialog control
    // Define the initial directory that is shown
    selectFileDialog.InitialDirectory = Application.StartupPath + @"Data";
    // Set the title of open file dialog
    selectFileDialog.Title = "Browse Txt Files";
    // DefaultExt is only used when "All Files" is selected
    // from the filter box and no extension is specified
    // by the user.
    selectFileDialog.DefaultExt = "txt";
    selectFileDialog.Filter = "Text Files (*.txt)|*.txt|All Files (*.*)|*.*";
}

```

Figure 12: Initialize File Dialog

- d. Configure the Event Handler.
  - a. Still in the form, be sure the button is selected and create a click event handler for the Read File button using the Events button and name the Click

"BtnReadFileClickEvent" as shown in Figure 13. Make sure we are using PascalCasing for all method names.

- 7. Add in the Method comments as shown in Figure 13.

```
/// <summary>
/// Click Event Handler to Read the File
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void BtnReadFileClickEvent(object sender, EventArgs e)
{
}
}
```

Figure 13: Comments for Event Handler Method

- 8. Whenever a new cs page is started, the first item we do is add the citations at the very top as shown in Figure 14.

```
activity 3
1  /*
2    * Your Name Here
3    * CST-150
4    * Project Name Here
5    * Date
6    * Citation(s) Here
7    */
8
9
10
11 namespace CST_150_Activity_3
12 {
}
```

Figure 14: Citation at the top.

- 9. Start coding the backend.
- 10. Open the FrmMain.cs file.

- 7. In the constructor, be sure the results label and selected file label are not visible as shown in Figure 15.
- 8. Run the code and verify everything is working and lblResults is not visible.

```
selectFileDialog.DefaultExt = ".txt";
selectFileDialog.Filter = "Text Files (*.txt)|*.txt|All Files (*.*)|*.*";

// When the form is initialized make sure the lblResults
// and lblSelectedFile are not visible
lblResults.Visible = false;
lblSelectedFile.Visible = false;
}
```

Figure 15: Hide Results Label

- 9. Add in the Declare and Initialize variables section as shown in Figure 16.

```
/// <param name="e" />
1 reference
private void BtnReadFileClickEvent(object sender, EventArgs e)
{
    // Declare and initialize variables
    string txtFile = "";
    string dirLocation = "";
    const int PadSpace = 20;

    // Once the button is clicked - show the file dialog
    if (this.selectFileDialog.ShowDialog() == DialogResult.OK)
    {
        // Read in the text file that was selected
    }
}
```

Figure 16: Declare and Initialize

- 10. Show the file dialog selection box when the button is clicked as shown in Figure 17.

```

/// <summary>
/// Click Event Handler to Read the File
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void BtnReadFileClickEvent(object sender, EventArgs e)
{
    // Once the button is clicked - show the file dialog
    if (this.selectFileDialog.ShowDialog() == DialogResult.OK)
    {
    }
}

```

Figure 17: Open File Dialog, so user can select a file to open.

- f. Run the program and verify the Select File Dialog box is visible when clicking on the button.

## 7. Configure the Text File

- a. Before we continue, we need to create the txt file.
- b. Go to the application directory "bin\Debug\net7.0-windows directory\Data" and create the data directory.
- c. Right click on the mouse and select new followed by text document.
- d. Name it "Topic3.txt" as shown in Figure 18.

<input type="checkbox"/> Name	Date modified	Type	Size
<input checked="" type="checkbox"/> Topic3.txt	2/7/2023 8:42 PM	Text Document	0 KB

Figure 18: Text File

- e. The instructor will provide the contents of this file. Be sure to use the provided content to receive full credit for this assignment.

## 8. Continue coding the back end.

- a. Follow the comments in the code to select the text file and show the selected file name and path as shown in Figure and Figure 20.

```

1 reference
private void BtnReadFileClickEvent(object sender, EventArgs e)
{
    // Declare and initialize variables
    string txtFile = "";
    string dirLocation = "";

    // Once the button is clicked - show the file dialog
    if (this.selectFileDialog.ShowDialog() == DialogResult.OK)
    {
        // Read in the text file that was selected
        txtFile = this.selectFileDialog.FileName;
        // Get the path of the file plus the filename
        dirLocation = Path.GetFullPath(selectFileDialog.FileName);
        // Show the selected file and path in the label
        lblSelectedFile.Text = txtFile;
        // Make sure to make this label visible
        lblSelectedFile.Visible = true;
    }
}

```

Figure 19: Get text file and show the file name.

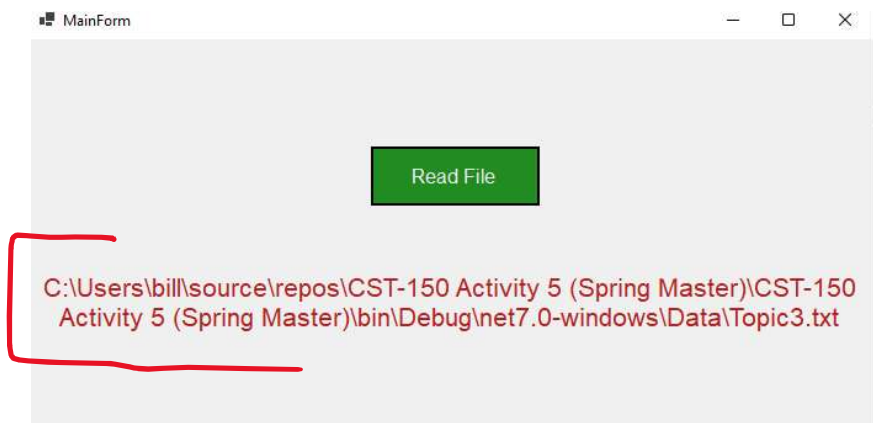


Figure 20: Example of how the display should look.

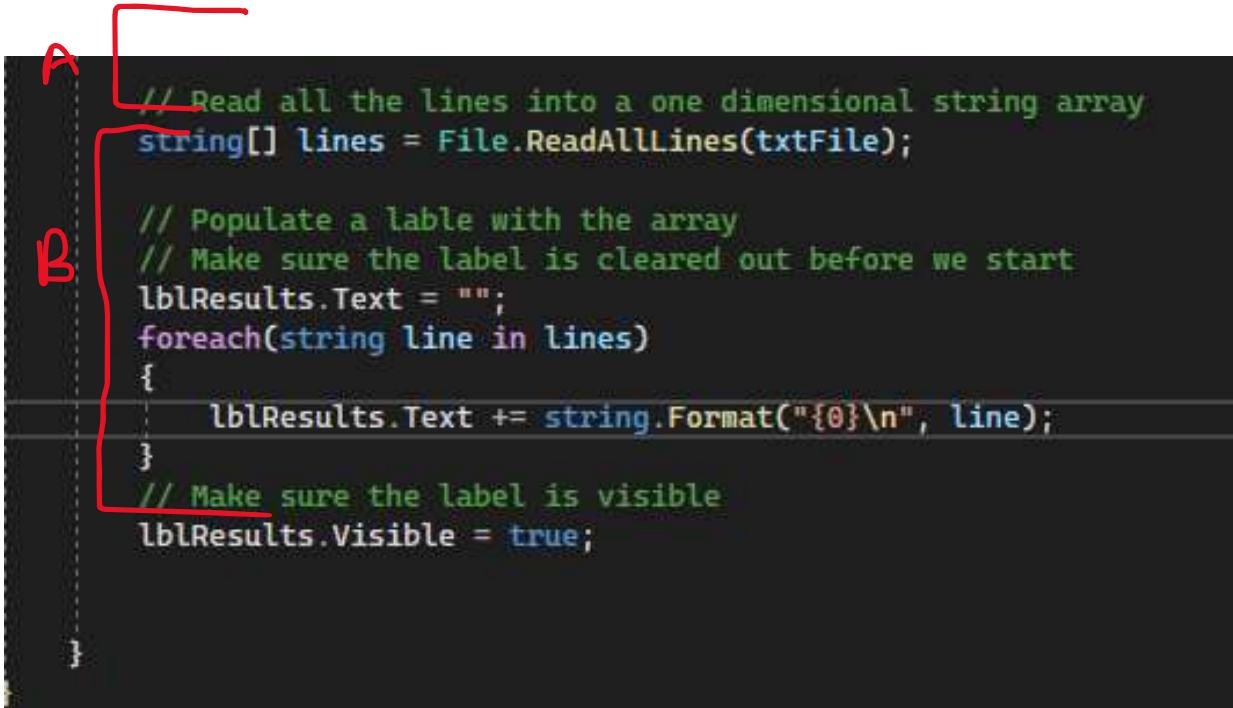
6. Now, we need to read in the contents of the file as shown in Figure 21 part A.

**Note:** This activity reads all lines of the text file in one statement and places the results inside a string array named lines. When working with large files, a software developer will typically read a text file one line at a time, which Activity 6 explains. However, it is important for a software developer to learn both methods of reading all lines first and then one line at a time.

7. Then, populate the lblResults as we iterate through the array using a foreach loop as shown in Figure 21 part B.



- Run the program. Notice how the data is not lined up very well.



```
// Read all the lines into a one dimensional string array
string[] lines = File.ReadAllLines(txtFile);

// Populate a table with the array
// Make sure the label is cleared out before we start
lblResults.Text = "";
foreach(string line in lines)
{
    lblResults.Text += string.Format("{0}\n", line);
}

// Make sure the label is visible
lblResults.Visible = true;
```

Figure 21: Read File

- Format the label.
  - a In order for each element to be perfectly lined up vertically, we must use a font that has equal spacing for each character.
  - b. Change the font for the lblResults to "Lucid Console" as shown in Figure 23.
  - Also, move the label to the left a little so everything will fit as shown in Figure 22.



Figure 22: Main Form

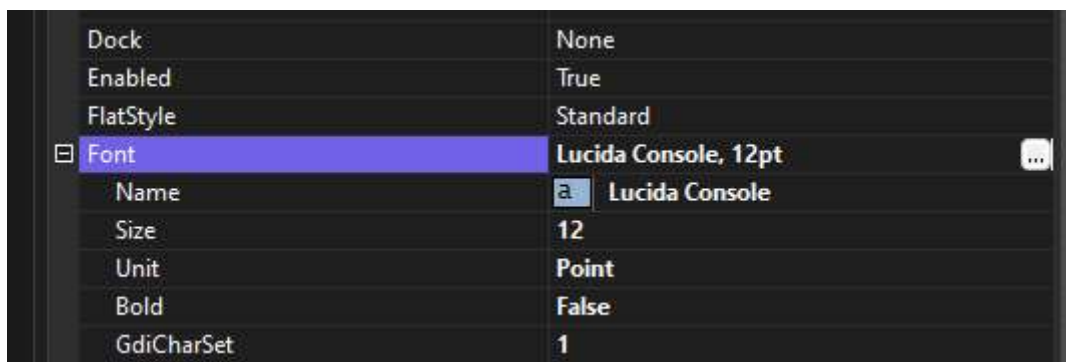


Figure 23: Label Font

- Now, iterate through the array and break each line into its own elements and then display the results as shown in Figure 24. This code replaces the "lblResults.Text += string.Format...." line in Figure 21.



```

string[] lines = File.ReadAllLines(txtFile);

// Populate a label with the array
// Make sure the label is cleared out before we start
lblResults.Text = "";
foreach(string line in lines)
{
    // Split each line into an array of elements
    string[] inventoryList = line.Split(", ");
    // Iterate through each element in the array
    // using a for loop instead of foreach loop
    for(int i = 0; i < inventoryList.Length; i++)
    {
        // Display each element using proper spacing
        lblResults.Text += inventoryList[i].PadRight(PadSpace);
    }
    // Need a new line after each iteration to show next line
    lblResults.Text += "\n";
}
// Make sure the label is visible
lblResults.Visible = true;

```

Figure 24: Format Results

10. Add in the Header Text.

- a. To make it more complete, add the header text.
- b. First add in the header and header lines as shown in Figure 25.

```

1 reference
private void BtnReadFileClickEvent(object sender, EventArgs e)
{
    // Declare and initialize variables
    string txtFile = "";
    string dirLocation = "";
    const int PadSpace = 20;
    string header1 = "Type", header2 = "Color", header3 = "Qty";
    string headerLine1 = "----", headerLine2 = "-----", headerLine3 = "----";

    // Once the button is clicked - show the file dialog
    if (this.selectFileDialog.ShowDialog() == DialogResult.OK)
    {

```

Figure 25: Header

- c. Then, add in the header to the label as shown in Figure 26.

```

string[] lines = File.ReadAllLines(txtFile);

// Populate a table with the array
// Make sure the label is cleared out before we start
lblResults.Text = "";
// Add in the header
lblResults.Text = string.Format("{0}{1}{2}\n", header1.PadRight(PadSpace), header2.PadRight(PadSpace), header3.PadRight(PadSpace));
lblResults.Text += string.Format("{0}{1}{2}\n", headerLine1.PadRight(PadSpace), headerLine2.PadRight(PadSpace), headerLine3.PadRight(PadSpace));
foreach (string line in lines)
{
    // Split each line into an array of elements
    string[] inventoryList = line.Split(", ");
    // Iterate through each element in the array

```

Figure 26: Header to Label

- 4. The display should look something like the following (keeping in mind the text file provided by the instructor will be different so the output text will be different) as shown in Figure 27.

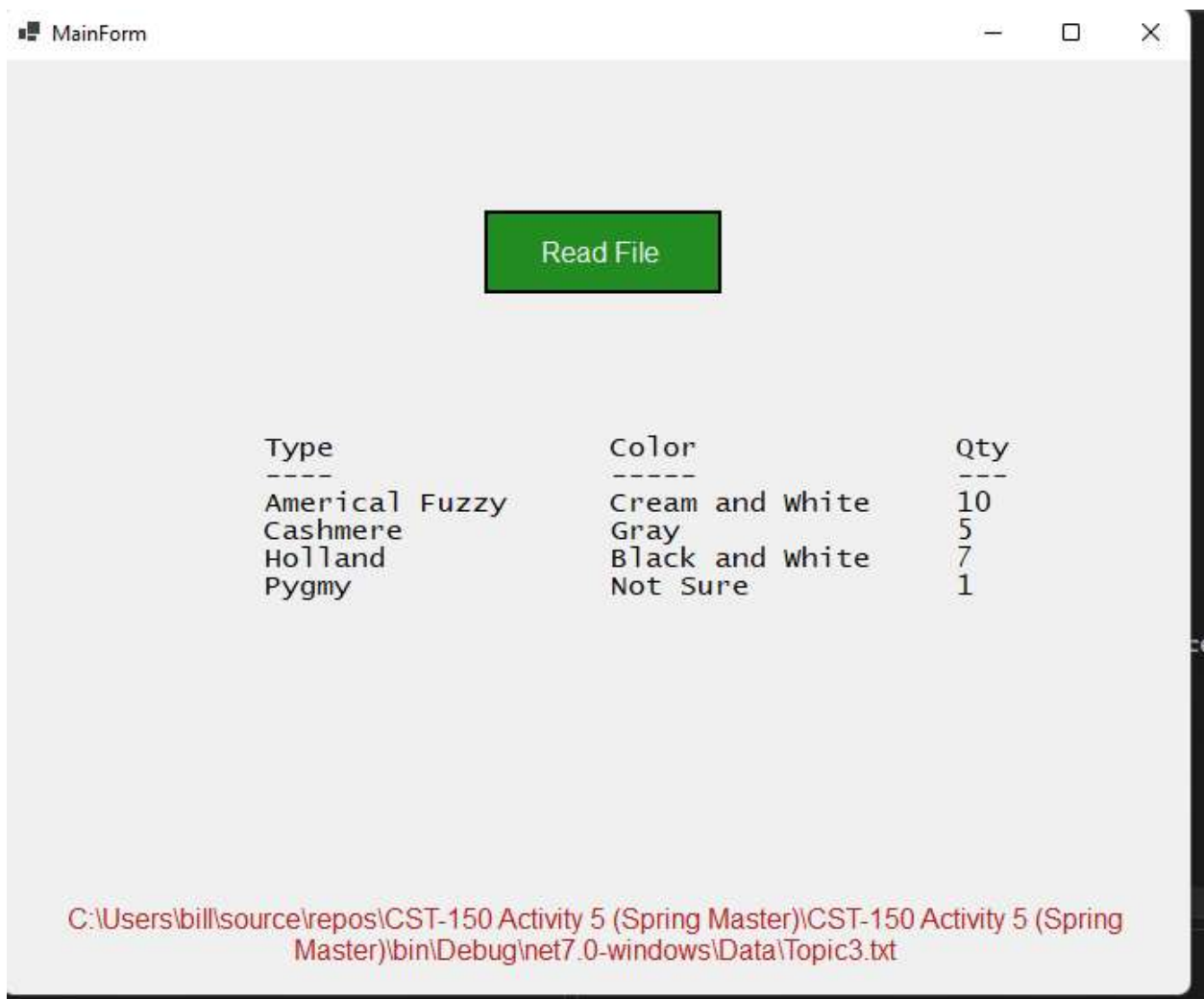


Figure 27: Final Result

- 1. Submit the Activity as described in the digital classroom.

- Be sure to use the text file that is provided by your instructor before taking screenshots.