COVER PAGE

Name: Eric Gathinji

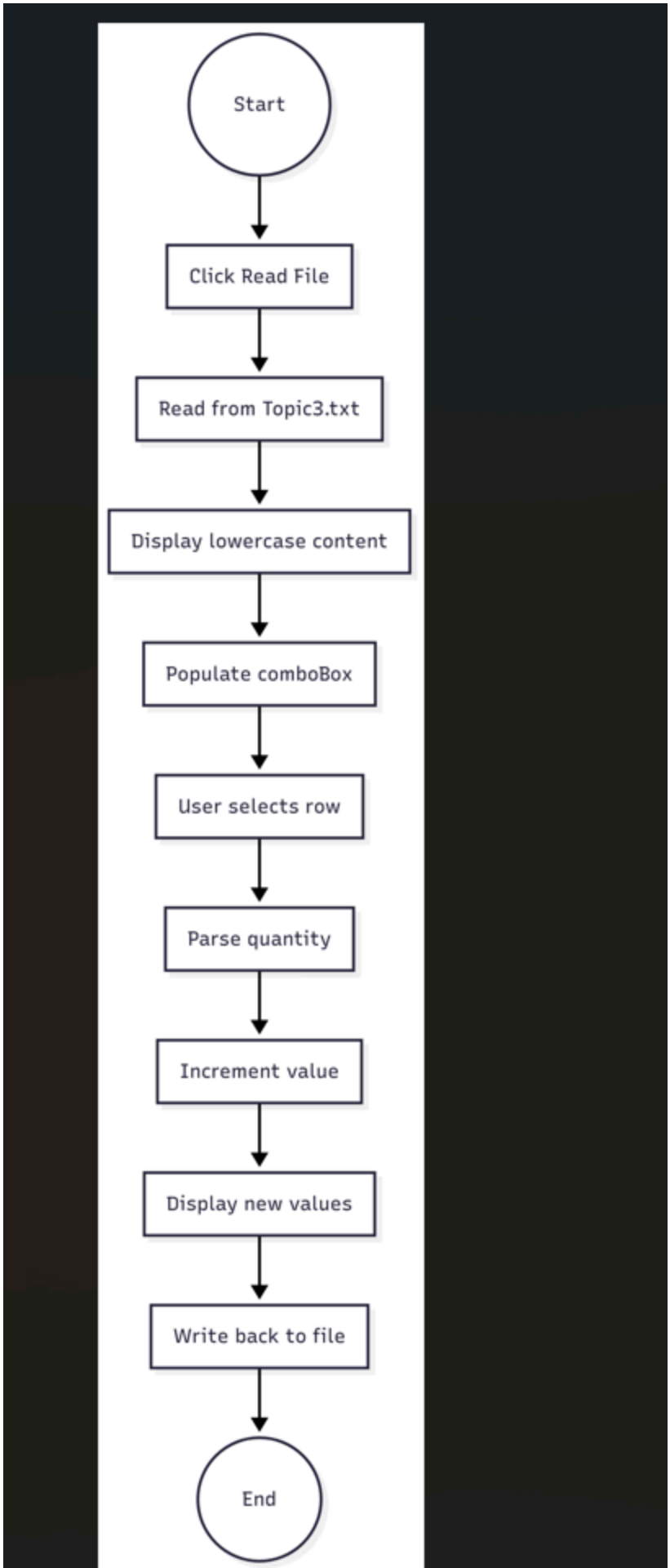
Programming in C# CST-150-0500

Grand Canyon University

5th July 2025

Activity 4

FLOWCHART

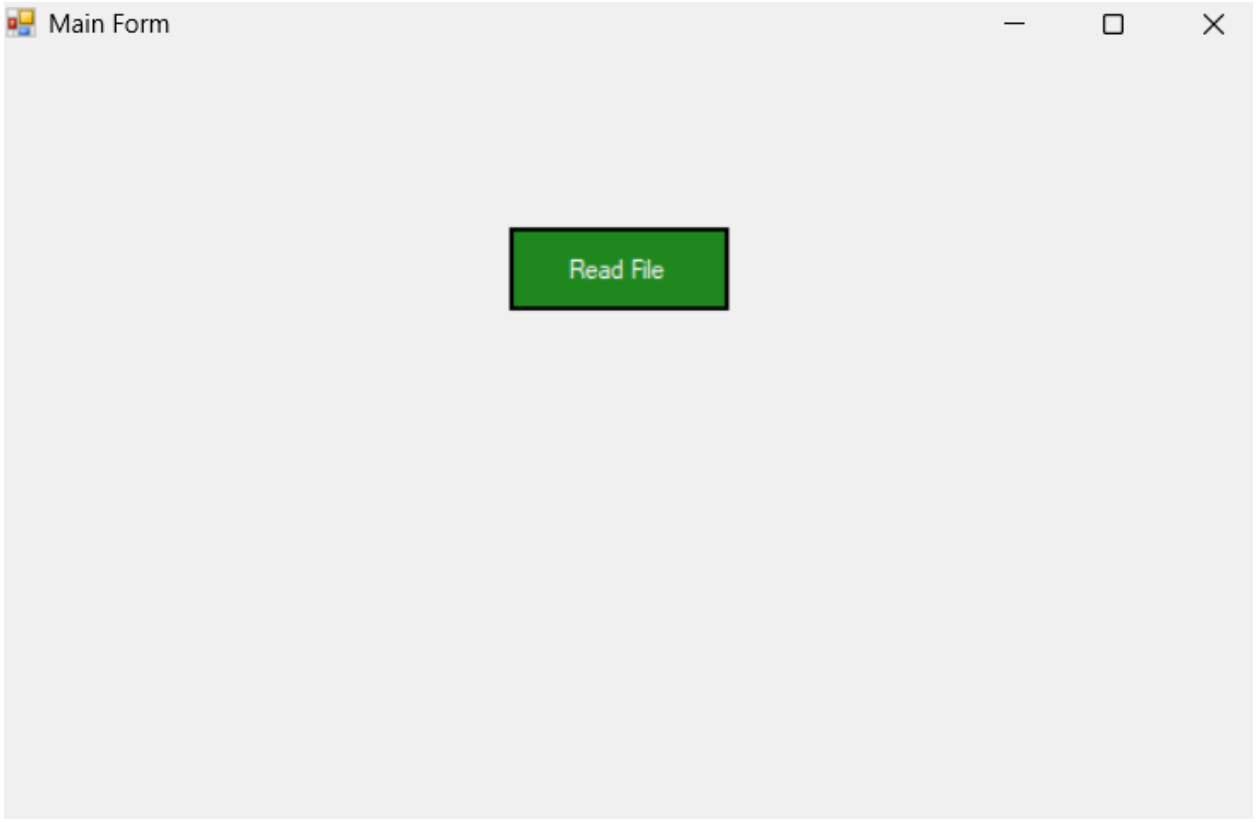

Start
Click Read File
Read from Topic3.txt
Display lowercase content
Populate comboBox
User selects row
Parse quantity
Increment value
Display new values
Write back to file
End

flow of the application, from reading the file to updating the quantity.

EXPLANATION;

This flowchart visualizes how the program works from start to finish.
It outlines the event-driven structure of the application triggered by user interaction.
Logic is modularized into methods to ensure the main form remains clean and readable.

Screenshot of the Form Before Populated

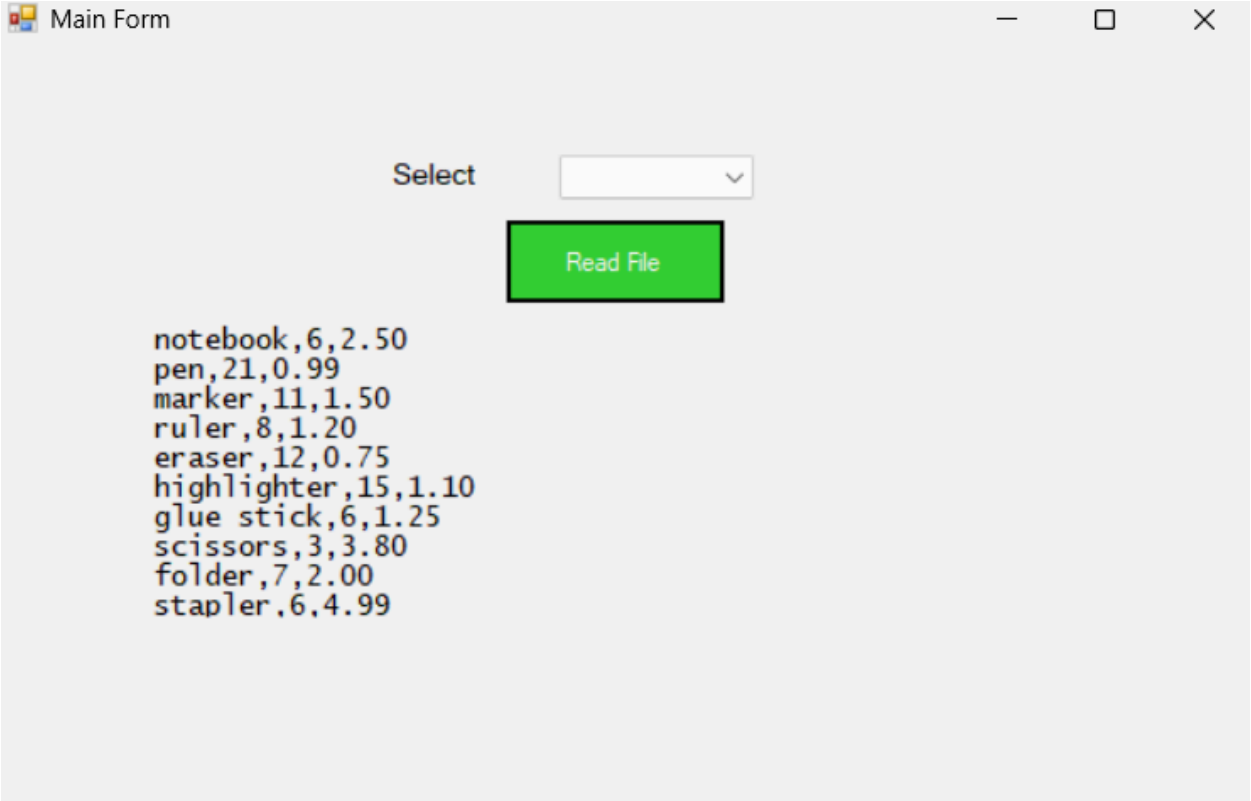


Initial form before loading file.

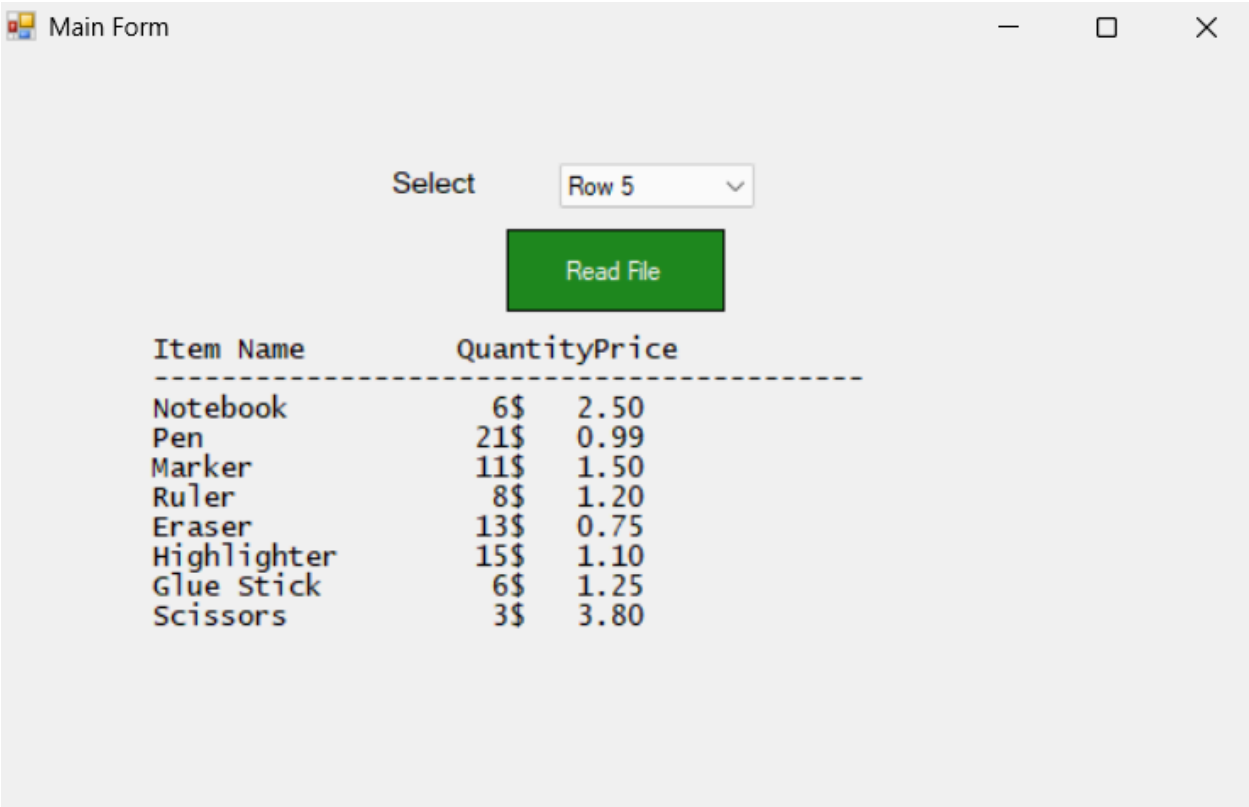
EXPLANATION;

The form starts with minimal visible components.
The label for results and the comboBox are hidden to reduce clutter.
Only the Read File button is active, prompting the user to load data.

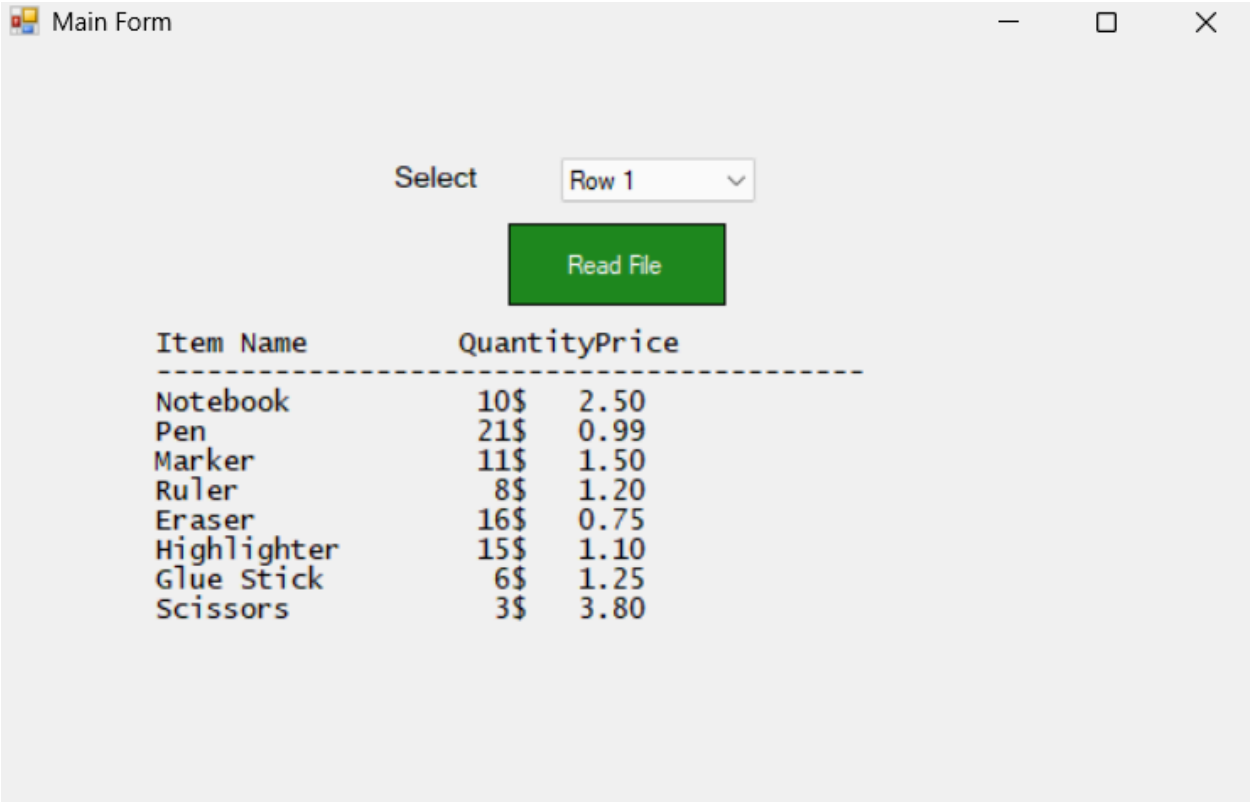
SCREENSHOTS OF THE FORMS AFTER BEING POPULATED;



Form after file is read and content converted to lowercase.



Form showing dropdown for row selection.



Form after quantity has been incremented.

EXPLANATION;
These screenshots demonstrate different phases of the program.
When the file is read, the content is formatted and converted to lowercase.
On selecting a row(1), the quantity is incremented and results are updated both visually and in the file.

Screenshot(s) of Code Behind

```
private void BtnReadFileClickEvent(object sender, EventArgs e)
{
    try
    {
        txtFilePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Data", "Topic3.txt");

        if (!File.Exists(txtFilePath))
        {
            MessageBox.Show("The file Topic3.txt was not found.");
            return;
        }

        lines = File.ReadAllLines(txtFilePath);

        // Convert content to lowercase and display
        string fileContent = string.Join(Environment.NewLine, lines);
        ConvertLowerCase(fileContent);

        // Populate comboBox with row numbers
        cmbSelectRow.Items.Clear();
        for (int i = 1; i <= lines.Length; i++)
        {
            cmbSelectRow.Items.Add($"Row {i}");
        }

        lblSelectRow.Visible = true;
        cmbSelectRow.Visible = true;
    }
    catch (Exception ex)
    {
        MessageBox.Show("An error occurred while reading the file:\n" + ex.Message);
```

```
/// Increments the quantity of the selected row by 1.
/// </summary>
1 reference
private void GetQty(int rowIndex)
{
    try
    {
        string[] splitLine = lines[rowIndex].Split(',');
        int quantity = int.Parse(splitLine[1]);
        quantity++;
        splitLine[1] = quantity.ToString();
        lines[rowIndex] = string.Join(",", splitLine);
    }
    catch (FormatException)
    {
        MessageBox.Show("Quantity could not be parsed to an integer.");
    }
}

/// <summary>
/// Writes updated data back to the text file.
/// </summary>
1 reference
private void IncDisplayQty()
{
    File.WriteAllLines(txtFilePath, lines);
}

/// <summary>
/// Handles comboBox selection to increment quantity.
/// </summary>
1 reference
private void SelectRowToInc(object sender, EventArgs e)
{
    int rowSelected = cmbSelectRow.SelectedIndex;
```

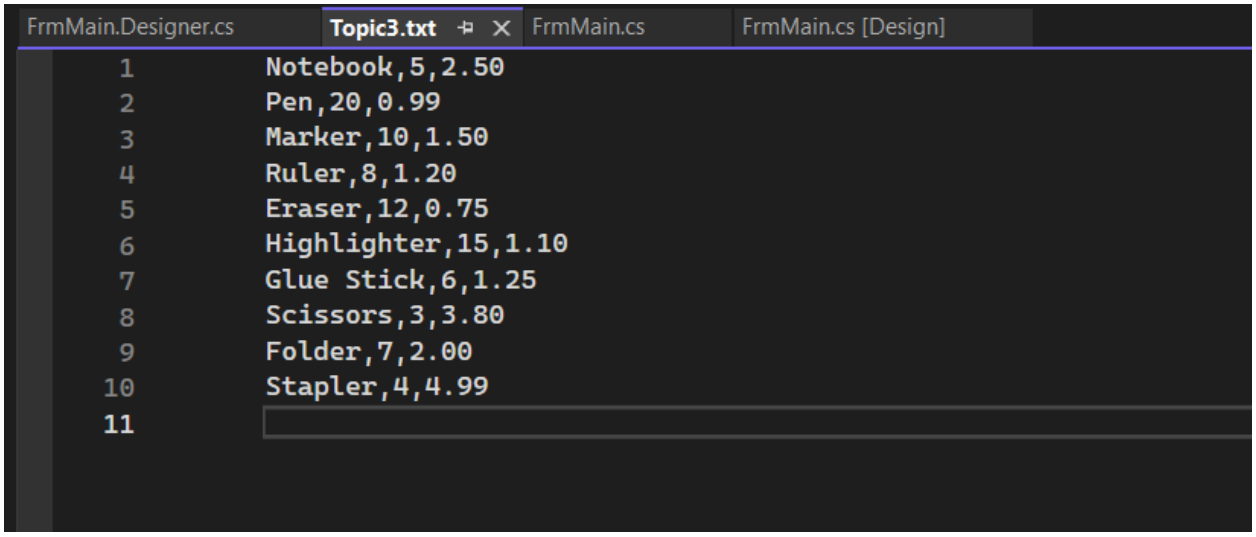
```
    1 reference
    private void IncDisplayQty()
    {
        File.WriteAllLines(txtFilePath, lines);
    }

    /// <summary>
    /// Handles comboBox selection to increment quantity.
    /// </summary>
    1 reference
    private void SelectRowToInc(object sender, EventArgs e)
    {
        int rowSelected = cmbSelectRow.SelectedIndex;
        if (rowSelected >= 0)
        {
            GetQty(rowSelected);
            IncDisplayQty();
            DisplayInventory(); // Refresh display
        }
        else
        {
            MessageBox.Show("Please select a valid row.");
        }
    }
}
```

Code-behind for reading and displaying file content.

EXPLANATION;

The code-behind is modularized into methods like `ConvertLowerCase` and `DisplayInventory`.
This approach ensures clarity and supports reusability.
Exception handling is used during parsing to avoid runtime errors.

Screenshot of Original Text File.

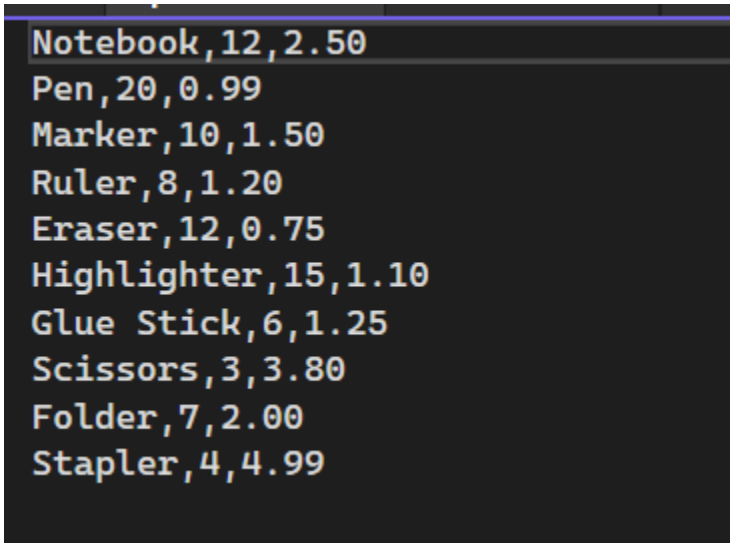


Original Topic3.txt file.

EXPLANATION;

The file contains inventory items with fields for name, quantity, and price separated by commas. This format is parsed line-by-line for processing within the program.

Screenshot of Updated Text File.



Updated Topic3.txt file after quantity increment.

EXPLANATION;

Only the selected row's quantity is modified, preserving all other entries.

This demonstrates that the program successfully writes changes back to the source file using `File.WriteAllLines`.