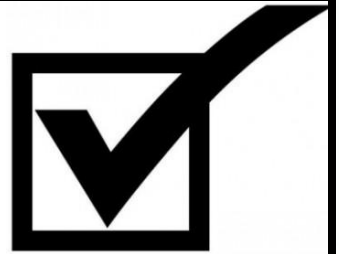


**BUSINESS
PROFESSIONALS
of AMERICA**
Giving Purpose to Potential



C# PROGRAMMING

(330)

REGIONAL 2026

PRODUCTION:

Regional_C_Sharp

_____ (1030 points)

Test Time: 90 minutes

GENERAL GUIDELINES.

Failure to follow any of these rules may result in disqualification:

1. **Submission Requirements:** Contestants must submit this test booklet along with any printouts.
2. **Permitted Items:** Only the equipment, supplies, and materials specified for this event are allowed in the testing area. Previous BPA tests and sample tests (whether handwritten, photocopied, or typed) are not permitted.
3. **Electronic Devices:** Electronic devices will be monitored according to ACT standards.

EXAM GUIDELINES.

1. Your name and/or school name should *not* appear on work you submit for grading.
2. Create a folder on the flash drive provided using your Contestant ID as the name of the folder.
3. Copy your entire solution/project into this folder. The project folder for you has already been provided: Regional_C_Sharp
4. Submit your entire solution/project so that the graders may open your project to review the source code.
5. Ensure that the files required to run your program are present and will execute on the flash drive provided.
6. You will need to use Visual Studios 2019 or greater to complete this exam.

*Note that the flash drive letter may *not* be the same when the program is graded as it was when you created the program.

The graders will *not* alter your source code. Submissions that do *not* contain source code will *not* be graded.

Windows Form1 Template Design

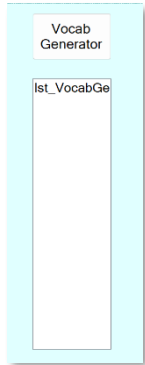
See the image below to familiarize yourself with the template of the Windows Form you will be programming. You will not be required to adjust any of the objects in the form, and all of the programmable event method signatures have been created for you (i.e. clicking on a button).

The image shows a Windows Form titled "Form1" with a standard Windows window border. The form is divided into several colored panels, each containing specific controls:

- Light Blue Panel (Top Left):** Contains a "Vocab Generator" button and a text box labeled "lst_VocabGe".
- Light Green Panel (Top Middle):** Contains a "Monthly Payment" button, a "Principal" text box, an "Interest Paid" text box, a "Time in Years" text box, and a "lst_MonthlyPayment" list box.
- Light Orange Panel (Top Right):** Contains a "Backwards" button, an "Enter Here" text box, a label "lbl_Backwards", a "No Vowels" button, another "Enter Here" text box, and a label "lbl_NoVowels".
- Red Panel (Top Far Right):** Contains an "Add Evens" button, an "Enter Here" text box, a "Digit Types" button, and another "Enter Here" text box.
- Light Blue Panel (Bottom Middle):** Contains a "Distance Converter" button, a text box, a "Convert To:" label, two radio buttons labeled "Miles" and "Kilometers", and a label "lst_Distance".
- Light Orange Panel (Bottom Left):** Contains a "Random Font" button, a label "Random Font!!!!", an "Add" button, a "Subtract" button, and a label "Add or Subtract".
- Light Gray Panel (Bottom Right):** Contains a "Hide or Show" button.
- Gray Panel (Top Far Right):** Contains a "Grader Button".

Regional C Sharp: The Buttons You Will Program:

1. Vocab Generator



Button Text: "Vocab Generator"

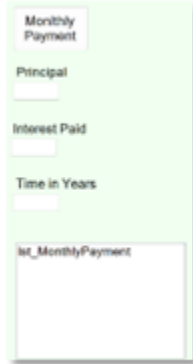
Instructions:

- Create a `List<string>` or equivalent data structure to store vocabulary terms like "Algorithm," "Binary," "Compiler," and so on.
- When the button is clicked:
 - Check if the associated `ListBox` is empty.
 - If it is empty, populate the `ListBox` with the vocabulary terms from the data structure.
 - If it is not empty, clear the `ListBox`.
 - The vocabulary words have already been provided for you.

private void btn_VocabGen_Click(object sender, EventArgs e)

- Handles the "Vocab Generator" button click event; toggles between populating and clearing a vocabulary list.

2. Monthly Payment



Button Text: "Monthly Payment"

Instructions:

- Take inputs from text boxes for:
 - **Principal** (Loan amount)
 - **Annual Interest Rate**
 - **Loan Term in Years**
- Convert the annual interest rate to a monthly rate and the loan term into months.

$$\text{Monthly Payment} = \frac{r \times P \times (1 + r)^n}{(1 + r)^n - 1}$$

- Use the provided financial formula to calculate the monthly payment:
- Where:
 - r = Monthly Interest Rate
 - P = Principal
 - n = Total Number of Payments
- Display the calculated monthly payment in the associated `ListBox`.
- Handle input errors by showing a message box if inputs are invalid.

private void btn_MonthlyPayment_Click(object sender, EventArgs e)

- Handles the "Monthly Payment" button click event; calculates and displays monthly payment based on loan details.

3. Backwards



Button Text: "Backwards"

Instructions:

- Retrieve the text input from the associated `TextBox`.
- Reverse the text.
- Display the reversed text in the associated `Label`.

private void btn_Backwards_Click(object sender, EventArgs e)

- Handles the "Backwards" button click event; reverses the input text and displays it.

4. No Vowels



- **Button Text:** "No Vowels"

Instructions:

- Retrieve the text input from the associated `TextBox`.
- Remove all vowels (a, e, i, o, u, in both uppercase and lowercase) from the text.
- Display the text without vowels in the associated `Label`.
- ***private void btn_NoVowels_Click(object sender, EventArgs e)***
- Handles the "No Vowels" button click event; removes vowels from input text and displays the result.

5. Add Evens



- **Button Text:** "Add Evens"

Instructions:

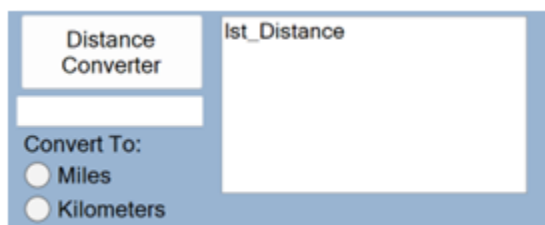
- Retrieve an integer input from the associated `TextBox`.
- Validate that the input is a positive integer greater than or equal to 2.
- Calculate the sum of all even numbers from 2 up to and including the input value.
- Display the sum in a message box.
- ***private void btn_AddEvens_Click(object sender, EventArgs e)***
- Handles the "Add Evens" button click event; calculates the sum of all even numbers up to a given value.

6. Digit Types



- **Button Text:** "Digit Types"
- **Instructions:**
- Retrieve the input text from the associated `TextBox`.
- Validate that the input contains valid numeric characters (integers or decimals).
- Count and classify the digits into categories:
 - Total digits
 - Even digits
 - Odd digits
 - Zero digits
- Display the counts in a message box.
- ***private void btn_Digits_Click(object sender, EventArgs e)***
- Handles the "Digit Types" button click event; analyzes input for total, even, odd, and zero digits.

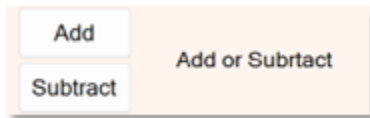
7. Distance Converter



-
- **Button Text:** "Distance Converter"
- **Instructions:**
- Retrieve a numeric input (distance) from the associated `TextBox`.
- Validate that the input is a positive number.
- Use the selected `RadioButton` to determine the conversion:
 - If converting **miles to kilometers**, multiply by 1.60934.
 - If converting **kilometers to miles**, multiply by 0.621371.
- Display the converted value and unit in the associated `ListBox`.
- ***private void btn_Distance_Click(object sender, EventArgs e)***

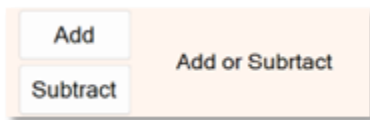
- Handles the "Distance Converter" button click event; converts a distance between miles and kilometers based on user input.

8. Add



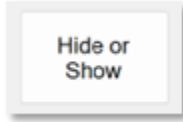
- **Button Text: "Add"**
Instructions:
 - Increment a counter variable each time the button is clicked.
 - Update the associated Label (`lbl_AddSubtract`) to display the current counter value.
 - Change the color of the Label:
 - Blue if the counter is zero or positive.
 - Red if the counter is negative.
- ***private void btn_Add_Click(object sender, EventArgs e)***
 - Handles the "Add" button click event; increments a counter and updates its display.
- ***private void UpdateCounterDisplay()***
 - Updates the display of the counter value and changes its color based on whether the value is positive, negative, or zero.

9. Subtract



- **Button Text: "Subtract"**
Instructions:
 - Decrement the counter variable each time the button is clicked.
 - Update the associated Label (`lbl_AddSubtract`) to display the current counter value.
 - Change the color of the Label:
 - Blue if the counter is zero or positive.
 - Red if the counter is negative.
- ***private void btn_Subtract_Click(object sender, EventArgs e)***
 - Handles the "Subtract" button click event; decrements a counter and updates its display.
- ***private void UpdateCounterDisplay()***
 - Updates the display of the counter value and changes its color based on whether the value is positive, negative, or zero.

10. Hide or Show



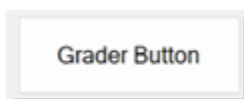
- **Button Text:** "Hide or Show"
- **Instructions:**
 - Toggle the visibility of the associated `Panel (pnl_HideShow)`:
 - If the panel is visible, hide it and update the button text to "Show Panel."
 - If the panel is hidden, show it and update the button text to "Hide Panel."
 - ***private void btn_HideShow_Click(object sender, EventArgs e)***
 - Handles the "Hide or Show" button click event; toggles the visibility of a specific panel and updates the button text.

11. Random Font



- **Button Text:** "Random Font"
- **Instructions:**
 - Generate a random font size between 8 and 20.
 - Randomly select a font style from available options such as:
 - Regular
 - Bold
 - Italic
 - Underline
 - Strikeout
 - Generate a random color by creating random RGB values.
 - Apply the random font size, style, and color to the associated `Label (lbl_RandomFont)`.
 - Ensure the changes are immediately visible to the user.
 - ***private void btn_RandomFont_Click(object sender, EventArgs e)***
 - Handles the "Random Font" button click event; assigns a random font size, style, and color to a label.
 -

12. Grader Button



- **Button Text:** "Grader Button"
- **Instructions:**
 - This button should only be worked on once you have ensured these items already work manually: `btn_VocabGen`, `btn_MonthlyPayment`, `btn_Backwards`,

btn_NoVowels, btn_AddEvens, btn_Digits, btn_Distance (km and Miles))

- Simulate pressing all the other buttons in the form programmatically:
 - Populate the vocabulary list.
 - Calculate a sample monthly payment.
 - Reverse a sample text.
 - Remove vowels from a sample text.
 - Calculate the sum of even numbers for a sample input.
 - Analyze digits for a sample numeric input.
 - Perform distance conversion for sample values (km and miles)
- Display a summary message indicating that all tests have been completed.
- ***private void btn_GraderButton_Click(object sender, EventArgs e)***
- Handles the "Grader Button" click event; performs tests on multiple other buttons and functionalities in the form.

Rubric:

Solution and Project (There is NO partial credit) (NOTE: UC represents uppercase and LC represents lowercase)		
The VS project file is present on the flash drive in a single folder with your contest ID		20 points
Button Execution (If the program does not execute, then the remaining items in this section receive a score of zero). The		
btn_VocabGen Vocab Generator		20 points
btn_MonthlyPayment Monthly Payment		80 points
btn_Backwards Backwards		30 points
btn_NoVowels No Vowels		40 points
btn_AddEvens Add Evens		50 points
btn_Digits Digit Types		70 points
btn_Distance Distance Converter		100 points
btn_Add Add		30 points
btn_Subtract Subtract		30 points
btn_HideShow Hide or Show		40 points
btn_RandomFont Random Font		30 points
btn_GraderButton Grader Button		200 points
(NOTE: you must test the following: btn_VocabGen, btn_MonthlyPayment, btn_Backwards, btn_NoVowels, btn_AddEvens, btn_Digits, btn_Distance (km and Miles)) (NO PARTIAL CREDIT)		
Invalid Input Handling Monthly Payment		10 points
Invalid Input Handling Add Evens		10 points
Invalid Input Handling Digit Types		10 points
Invalid Input Handling Distance Converter		10 points
Subtotal		/780 Points
Source Code Review (Code must work to get credit.)		
Program is properly commented (NOTE: in order to get the full credit of 50 points for the commenting requirement, you should place the comment flag in front of the comment in your code. The comment flag will precede the explanation. For example, if the flag is SC1, your comment should read as "//SC1..." in front of the part of the code being reviewed. Partial Credit may be awarded if some comments are present.)		50 points
A comment containing the contestant number is present at the top of the Form1.cs file		10 points
SC1: Vocab Generator: <i>private void btn_VocabGen_Click(object sender, EventArgs e)</i> Code that stores the list of Strings and populates a list box		10 points
SC2: Monthly Payment: <i>private void btn_MonthlyPayment_Click(object sender, EventArgs e)</i> Applying a mathematical formula for financial calculations and understanding order of operations.		30 points

SC3: Backwards: <i>private void btn_Backwards_Click(object sender, EventArgs e)</i> Code that uses a data structure to help reverse the string.		10 points
SC4: No Vowels: <i>private void btn_NoVowels_Click(object sender, EventArgs e)</i> Code that demonstrates filtering vowels from a string		10 points
SC5: Add Evens: <i>private void btn_AddEvens_Click(object sender, EventArgs e)</i> Creating a loop to iterate over even numbers and accumulating a sum.		10 points
SC6: Digit Types: <i>private void btn_Digits_Click(object sender, EventArgs e)</i> Detecting and categorizing digits using conditional logic.		10 points
SC7: Distance Converter: <i>private void btn_Distance_Click(object sender, EventArgs e)</i> Branching logic based on radio button selection.		10 points
SC8: Add: <i>private void btn_Add_Click(object sender, EventArgs e)</i> Incrementing a variable and calling a method to update the UI using <i>private void UpdateCounterDisplay()</i>		10 points
SC9: Subtract: <i>private void btn_Subtract_Click(object sender, EventArgs e)</i> Decrementing a variable and calling a method to update the UI. <i>private void UpdateCounterDisplay()</i>		10 points
SC10: Hide or Show: <i>private void btn_HideShow_Click(object sender, EventArgs e)</i> Toggling the visibility of a UI component.		10 points
SC11: Random Font: <i>private void btn_RandomFont_Click(object sender, EventArgs e)</i> Dynamically creating and applying a new font with specific attributes.		20 points
SC12 (must be placed in these 7 locations; no partial credit): Grader Button: <i>private void btn_GraderButton_Click(object sender, EventArgs e)</i> Programmatically triggering button click events to automate testing. It must test the following: btn_VocabGen, btn_MonthlyPayment, btn_Backwards, btn_NoVowels, btn_AddEvens, btn_Digits, btn_Distance (km and Miles)		50 points
Subtotal		/250 Points
Total Points		/1030

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Threading.Tasks;
9 using System.Windows.Forms;
10
11 namespace Regional_C_Sharp
12 {
13     3 references
14     public partial class Form1 : Form
15     {
16         private Random random = new Random();
17         private int counter = 0; // Counter variable to track the value
18         1 reference
19         public Form1()
20         {
21             InitializeComponent();
22         }
23         //SC1
24         1 reference
25         private void btn_VocabGen_Click(object sender, EventArgs e)
26         {
27             // Check if the ListBox is empty
28             if (lst_VocabGen.Items.Count == 0)
29             {
30                 // If empty, populate it with vocabulary terms
31                 List<string> vocabTerms = new List<string>
32                 {
33                     "Algorithm", "Binary", "Compiler", "Data Structure", "Encapsulation",
34                     "Function", "Inheritance", "Loop", "Object-Oriented Programming",
35                     "Parameter", "Queue", "Recursion", "Syntax", "Variable", "While Loop"
36                 };
37                 // Add each term to the ListBox
38                 lst_VocabGen.Items.AddRange(vocabTerms.ToArray());
39             }
40             else
41             {
42                 // If it already has items, clear the ListBox
43                 lst_VocabGen.Items.Clear();
44             }
45         }
46         //SC2
47         1 reference
48         private void btn_MonthlyPayment_Click(object sender, EventArgs e)
49         {
50             try
51             {
52                 // Parse the principal (P), annual interest rate, and years
53                 double principal = double.Parse(txt_Princ.Text);
54                 double annualRate = double.Parse(txt_Interest.Text);
55                 int years = int.Parse(txt_Years.Text);
56
57                 // Convert annual interest rate to monthly rate and years to months
58                 double monthlyRate = annualRate / 100 / 12;
59                 int months = years * 12;
60
61                 // Calculate the monthly payment using the formula
62                 double monthlyPayment = (monthlyRate * Math.Pow(1 + monthlyRate, months)) /
63                     (Math.Pow(1 + monthlyRate, months) - 1) * principal;
64
65                 // Clear the ListBox and display the result
66                 lst_MonthlyPayment.Items.Clear();
67                 lst_MonthlyPayment.Items.Add($"Monthly Payment: {monthlyPayment:C}");
68             }
69             catch (Exception ex)
70             {
71                 // Display an error message if parsing or calculation fails
72                 MessageBox.Show("Please enter valid numerical values for Principal, Interest, and Years.", "Input Error");
73             }
74         }
75     }
76 }
```

```
72 //SC3
73 1 reference
74 private void btn_Backwards_Click(object sender, EventArgs e)
75 {
76     string inputWord = txt_Backwards.Text;
77
78     // Reverse the word
79     char[] charArray = inputWord.ToCharArray();
80     Array.Reverse(charArray);
81     string reversedWord = new string(charArray);
82
83     // Display the reversed word in the label
84     lbl_Backwards.Text = reversedWord;
85 }
86 //SC4
87 1 reference
88 private void btn_NoVowels_Click(object sender, EventArgs e)
89 {
90     // Get the word entered in the textbox
91     string inputWord = txt_NoVowels.Text;
92
93     // Remove vowels (a, e, i, o, u, A, E, I, O, U) from the word
94     string noVowelsWord = new string(inputWord.Where(c => !"aeiouAEIOU".Contains(c)).ToArray());
95
96     // Display the result without vowels in the label
97     lbl_NoVowels.Text = noVowelsWord;
98 }
99 //SC5
100 1 reference
101 private void btn_AddEvens_Click(object sender, EventArgs e)
102 {
103     if (int.TryParse(txt_AddEvens.Text, out int maxValue))
104     {
105         // Check if the value is positive
106         if (maxValue >= 2)
107         {
108             int sum = 0;
109
110             // Add all even numbers from 2 up to and including maxValue
111             for (int i = 2; i <= maxValue; i += 2)
112             {
113                 sum += i;
114             }
115
116             // Display the result in a MessageBox or any label if needed
117             MessageBox.Show($"The sum of even numbers up to {maxValue} is: {sum}", "Sum of Evens");
118         }
119         else
120         {
121             // Display an error if the number is negative or less than 2
122             MessageBox.Show("Please enter a positive integer greater than or equal to 2.", "Input Error");
123         }
124     }
125     else
126     {
127         // Display an error if the input is not a valid integer
128         MessageBox.Show("Please enter a valid integer.", "Input Error");
129     }
130 }
131 //SC6
132 1 reference
133 private void btn_Digits_Click(object sender, EventArgs e)
134 {
135     // Retrieve the input from the textbox
136     string input = txt_Digits.Text;
137
138     // Attempt to parse the input as a double to allow integers and decimal values
139     if (double.TryParse(input, out _))
140     {
141         int totalDigits = 0;
142         int evenDigits = 0;
143         int oddDigits = 0;
144         int zeroDigits = 0;
```

```
141
142 // Iterate through each character in the input string
143 foreach (char c in input)
144 {
145     // Check if the character is a digit
146     if (char.IsDigit(c))
147     {
148         totalDigits++; // Count total digits
149
150         int digit = c - '0'; // Convert character to integer
151
152         // Classify the digit as even, odd, or zero
153         if (digit == 0)
154         {
155             zeroDigits++;
156         }
157         else if (digit % 2 == 0)
158         {
159             evenDigits++;
160         }
161         else
162         {
163             oddDigits++;
164         }
165     }
166 }
167
168 // Display the results in a message box
169 MessageBox.Show(
170     $"Total digits: {totalDigits}\n" +
171     $"Even digits: {evenDigits}\n" +
172     $"Odd digits: {oddDigits}\n" +
173     $"Zero digits: {zeroDigits}",
174     "Digit Analysis"
175 );
176 }
177 else
178 {
179     // Display an error message if the input is not a valid number
180     MessageBox.Show("Please enter a valid number (integer or decimal).", "Input Error");
181 }
182 }
183 //SC7
184 private void btn_Distance_Click(object sender, EventArgs e)
185 {
186     // Retrieve the input from the textbox
187     string input = txt_Distance.Text;
188
189     // Validate that the input is a positive numeric value
190     if (double.TryParse(input, out double distance) && distance > 0)
191     {
192         double convertedDistance;
193         string unit;
194
195         // Check if the Kilometers radio button is selected
196         if (rdb_KM.Checked)
197         {
198             // Convert miles to kilometers (1 mile ≈ 1.60934 kilometers)
199             convertedDistance = distance * 1.60934;
200             unit = "kilometers";
201         }
202         // Check if the Miles radio button is selected
203         else if (rdb_Miles.Checked)
204         {
205             // Convert kilometers to miles (1 kilometer ≈ 0.621371 miles)
206             convertedDistance = distance * 0.621371;
207             unit = "miles";
208         }
209         else
210         {
211             // If no radio button is selected, show an error message and exit
212             MessageBox.Show("Please select a unit to convert to (Kilometers or Miles).", "Selection Error");
213             return;
214         }
215     }
216 }
```

```
215 // Display the converted distance in the list box with unit
216 lst_Distance.Items.Add($"{distance} converted to {convertedDistance:F2} {unit}");
217 }
218 else
219 {
220 // Display an error message if the input is not valid
221 MessageBox.Show("Please enter a positive numeric value.", "Input Error");
222 }
223 }
224 }
225 //SC12
226 1 reference
227 private void btn_GraderButton_Click(object sender, EventArgs e)
228 {
229 // Test btn_VocabGen
230 btn_VocabGen.PerformClick();
231
232 // Test btn_MonthlyPayment
233 txt_Prin.Text = "10000"; // Set principal
234 txt_Interest.Text = "5"; // Set annual interest rate
235 txt_Years.Text = "10"; // Set loan term in years
236 btn_MonthlyPayment.PerformClick();
237
238 // Test btn_Backwards
239 txt_Backwards.Text = "example";
240 btn_Backwards.PerformClick();
241 MessageBox.Show($"Reversed Word: {lbl_Backwards.Text}", "Backwards Test");
242
243 // Test btn_NoVowels
244 txt_NoVowels.Text = "example";
245 btn_NoVowels.PerformClick();
246 MessageBox.Show($"Without Vowels: {lbl_NoVowels.Text}", "No Vowels Test");
247
248 // Test btn_AddEvens
249 txt_AddEvens.Text = "10";
250 btn_AddEvens.PerformClick();
251
252 // Test btn_Digits
253 txt_Digits.Text = "1234.56";
254 btn_Digits.PerformClick();
255
256 // Test btn_Distance
257 txt_Distance.Text = "10"; // Set distance
258 rdb_KM.Checked = true; // Select kilometers
259 btn_Distance.PerformClick();
260
261 rdb_Miles.Checked = true; // Change to miles and test again
262 btn_Distance.PerformClick();
263
264 MessageBox.Show("All buttons have been tested.", "Grader Button Test");
265 }
266 //SC11
267 1 reference
268 private void btn_RandomFont_Click(object sender, EventArgs e)
269 {
270 // Random font size between 8 and 20
271 float fontSize = random.Next(8, 21);
272
273 // Random font style
274 FontStyle[] styles = { FontStyle.Regular, FontStyle.Bold, FontStyle.Italic, FontStyle.Underline, FontStyle.Strikeout };
275 FontStyle randomStyle = styles[random.Next(styles.Length)];
276
277 // Create the new font
278 Font randomFont = new Font(lbl_RandomFont.Font.FontFamily, fontSize, randomStyle);
279
280 // Random color
281 Color randomColor = Color.FromArgb(random.Next(256), random.Next(256), random.Next(256));
282
283 // Apply random font and color to the label
284 lbl_RandomFont.Font = randomFont;
285 lbl_RandomFont.ForeColor = randomColor;
286 }
```



```
285 //SC8
286 1 reference
287 private void btn_Add_Click(object sender, EventArgs e)
288 {
289     counter++; // Increment the counter
290     UpdateCounterDisplay(); // Update the label with the new value and color
291 }
292 //SC9
293 1 reference
294 private void btn_Subtract_Click(object sender, EventArgs e)
295 {
296     counter--; // Decrement the counter
297     UpdateCounterDisplay(); // Update the label with the new value and color
298 }
299 2 references
300 private void UpdateCounterDisplay()
301 {
302     lbl_AddSubtract.Text = counter.ToString(); // Update the label text
303
304     // Set the label color based on the counter value
305     if (counter < 0)
306     {
307         lbl_AddSubtract.ForeColor = Color.Red; // Red for negative values
308     }
309     else
310     {
311         lbl_AddSubtract.ForeColor = Color.Blue; // Blue for zero or positive values
312     }
313 }
314 //SC10
315 1 reference
316 private void btn_HideShow_Click(object sender, EventArgs e)
317 {
318     // Toggle the visibility of the panel
319     pnl_HideShow.Visible = !pnl_HideShow.Visible;
320
321     // Optionally, update the button text to reflect the current state
322     if (pnl_HideShow.Visible)
323     {
324         btn_HideShow.Text = "Hide Panel"; // Update button text
325     }
326     else
327     {
328         btn_HideShow.Text = "Show Panel"; // Update button text
329     }
330 }
```