Lab 3: Declaring and Calling Methods

Exercise 1: Calculating the Greatest Common Divisor of Two Integers by Using Euclid's Algorithm

Task 1: Open the starter project

1. Log on to the 10266A-GEN-DEV virtual machine as **Student** with the password **Pa$$w0rd**.
2. Open Microsoft® Visual Studio® 2010:
   - Click **Start**, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Import the code snippets from the E:\Labfiles\Lab 3\Snippets folder:
   a. In Visual Studio, on the **Tools** menu, click **Code Snippets Manager**.
   b. In the **Code Snippets Manager** dialog box, click **Add**.
   c. In the **Code Snippets Directory** dialog box, move to the **E:\Labfiles\Lab 3\Snippets** folder, and then click **Select Folder**.
   d. In the **Code Snippets Manager** dialog box, click **OK**.
4. Open the Euclid solution in the E:\Labfiles\Lab 3\Ex1\Starter folder:
   a. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
   b. In the **Open Project** dialog box, move to the **E:\Labfiles\Lab 3\Ex1\Starter** folder, click **Euclid.sln**, and then click **Open**.

Task 2: Implement Euclid's algorithm

1. Review the task list:
   a. If the task list is not already visible, on the **View** menu, click **Task List**.
   b. If the **Task List** is displaying **User Tasks**, in the drop-down list box click **Comments**.

2. Use the Task List window to navigate to the **TODO Exercise 1, Task 2** task.

   This task is located in the GCDAlgorithms.cs file:

   a. If the task list is not already visible, on the **View** menu, click **Task List**.

   b. If the **Task List** is displaying **User Tasks**, in the drop-down list box click **Comments**.

   c. In the Task List window, double-click **TODO Exercise 1, Task 2**.

3. In the **GCDAlgorithms** class, remove the **TODO Exercise 1, Task 2** comment and declare a **public static** method called **FindGCDEuclid**. The method should accept two integer parameters called a and b, and return an integer value.

   Your code should resemble the following code example.

```
...
static class GCDAlgorithms
{
    public static int FindGCDEuclid(int a, int b)
    {

    }
}
...
```

4. In the **FindGCDEuclid** method, add code that calculates and returns the greatest common divisor (GCD) of the values specified by the parameters a and b by using Euclid's algorithm.

   Euclid's algorithm works as follows:

   a. If *a* is zero, the GCD of *a* and *b* is *b*.

   b. Otherwise, repeatedly subtract *b* from *a* (when *a* is greater than *b*) or subtract *a* from *b* (when *b* is greater than *a*) until *b* is zero.

   c. The GCD of the two original parameters is the new value in *a*.

   Your code should resemble the following code example.

```
...
static class GCDAlgorithms
{
    public static int FindGCDEuclid(int a, int b)
    {
        if (a == 0) return b;
        while (b != 0)
```

```
        {

            if (a > b)
            {
                a = a - b;
            }

            else
            {
                b = b - a;
            }

        }

        return a;
    }

}
...
```

Task 3: Test the FindGCDEuclid method

1. Use the Task List window to navigate to the **TODO Exercise 1, Task 3** task.

   This task is located in the MainWindow.xaml.cs file. This is the code-behind file for a Windows® Presentation Foundation (WPF) window that you will use to test the **FindGCDEuclid** method and display the results:

   - In the Task List window, double-click **TODO Exercise 1, Task 3**.

2. Remove the **TODO Exercise 1, Task 3** comment, add code to call the **static FindGCDEuclid** method of the **GCDAlgorithms** class, and display the results in the **resultEuclid** label control. In the method call, use the firstNumber and secondNumber variables as arguments (these variables contain values that the user enters in the WPF window). Finally, the result should be formatted as the following code example shows.

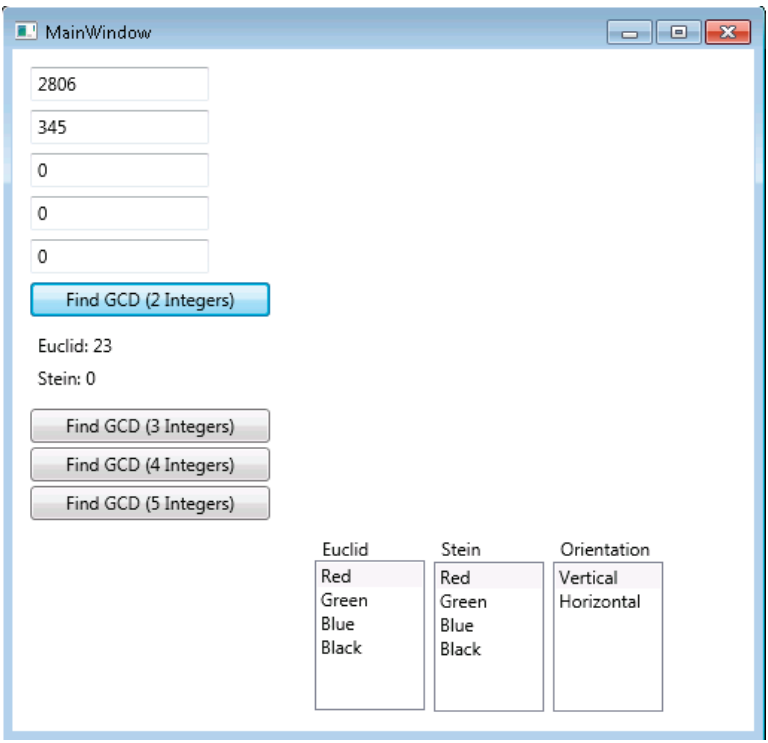
```
Euclid: result
```

Hint: Set the **Content** property of a label control to display data in a label. Use the **String.Format** method to create a formatted string.

Your code should resemble the following code example.

```
...
if (sender == findGCD) // Euclid for two integers
{
    // Invoke the FindGCD method and display the result
    this.resultEuclid.Content =
        String.Format("Euclid: {0}",
        GCDAlgorithms.FindGCDEuclid(firstNumber, secondNumber));
}
...
```

3. Build the solution and correct any errors:
   - On the **Build** menu, click **Build Solution**. Correct any errors.
4. Run the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Start Debugging**.
5. In the GreatestCommonDivisor application, in the MainWindow window, in the first text box, type **2806**
6. In the second text box, type **345** and then click **Find GCD (2 Integers)**. The result of **23** should be displayed, as the following screen shot shows.



7. Use the window to calculate the GCD for the values that are specified in the following table, and verify that the results that are displayed match those in the table.

First number	Second number	Result
0	0	0
0	10	10
25	10	5
25	100	25
26	100	2
27	100	1

8. Close the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Stop Debugging**.

Task 4: Create a unit test for the FindGCDEuclid method

1. Open the GCDAlgorithms.cs file:
   - In Solution Explorer, double-click **GCDAlgorithms.cs**.
2. In the **GCDAlgorithms** class, create a unit test for the **FindGCDEuclid** method. Create a new Test Project called GCD Test Project to hold the unit test:
   a. In the **GCDAlgorithms** class, right-click the **FindGCDEuclid** method, and then click **Create Unit Tests**.
   b. In the **Create Unit Tests** dialog box, ensure that the **FindGCDEuclid(System.Int32, System.Int32)** check box is selected, ensure that the **Output project** is set to **Create a new Visual C# test project**, and then click **OK**.
   c. In the **New Test Project** dialog box, in the **Enter a name for your new project**, type **GCD Test Project** and then click **Create**.
   d. If the **You have made changes to your tests** dialog box is displayed, click **OK**.

e. In the **Add InternalsVisibleTo Attribute** dialog box, click **Yes**.

3. In the GCD Test Project project, in the GCDAlgorithmsTest.cs file, locate the **FindGCDEuclidTest** method:

   - In the Code Editor window, in the **GCDAlgorithmsTest** class, locate the **FindGCDEuclidTest** method.

4. In the **FindGCDEuclidTest** method, set the a variable to **2806**, set the b variable to **345**, set the expected variable to **23**, and then remove the **Assert.Inconclusive** method call.

   Your code should resemble the following code example.

```
...

[TestMethod()]

public void FindGCDEuclidTest()

{
    int a = 2806; // TODO: Initialize to an appropriate value
    int b = 345; // TODO: Initialize to an appropriate value
    int expected = 23; // TODO: Initialize to an appropriate value
    int actual;
    actual = GCDAlgorithms.FindGCDEuclid(a, b);
    Assert.AreEqual(expected, actual);
}

...
```

5. Open the Test View window and refresh the display if the unit test is not listed:

   a. On the **Test** menu, point to **Windows**, and then click **Test View**.

   b. If the **You have made changes to your tests** dialog box appears, click **OK**.

   c. In the Test View window, click the **Refresh** button.

6. Run the FindGCDEuclidTest test and verify that the test ran successfully:

   a. In the Test View window, right-click **FindGCDEuclidTest**, and then click **Run Selection**.

   b. In the Test Results window, verify that the **FindGCDEuclidTest** test passed.

Exercise 2: Calculating the GCD of Three, Four, or Five Integers

Task 1: Open the starter project

- Open the Euclid solution in the E:\Labfiles\Lab 3\Ex2\Starter folder.

  This solution contains a completed copy of the code from Exercise 1:

  a. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.

  b. In the **Open Project** dialog box, move to the **E:\Labfiles\Lab 3\Ex2\Starter** folder, click **Euclid.sln**, and then click **Open**.

Task 2: Add overloaded methods to the GCDAlgorithms class

1. In Visual Studio, review the task list:

   a. If the task list is not already visible, on the **View** menu, click **Task List**.

   b. If the **Task List** is displaying **User Tasks**, in the drop-down list box click **Comments**.

2. Use the Task List window to navigate to the **TODO Exercise 2, Task 2** task:

   - In the Task List window, double-click **TODO Exercise 2, Task 2**.

3. In the **GCDAlgorithms** class, remove the **TODO Exercise 2, Task 2** comment, and then declare an overloaded version of the **FindGCDEuclid** method. The method should accept three integer parameters called a, b, and c, and return an integer value.

   Your code should resemble the following code example.

```
...
static class GCDAlgorithms
{
    ...
    public static int FindGCDEuclid(int a, int b, int c)
    {

    }
}
...
```

4. In the new method, add code that uses the original **FindGCDEuclid** method to find the GCD for the parameters a and b. Store the result in a new variable called d.

   Your code should resemble the following code example.

```
...
public static int FindGCDEuclid(int a, int b, int c)
{
    int d = FindGCDEuclid(a, b);
}
...
```

5. Add a second call to the original **FindGCDEuclid** method to find the GCD for variable d and parameter c. Store the result in a new variable called e.

   Your code should resemble the following code example.

```
...
public static int FindGCDEuclid(int a, int b, int c)
{
    int d = FindGCDEuclid(a, b);
    int e = FindGCDEuclid(d, c);
}
...
```

6. Add code to return the parameter e from the **FindGCDEuclid** method.

   Your code should resemble the following code example.

```
...
public static int FindGCDEuclid(int a, int b, int c)
{
    int d = FindGCDEuclid(a, b);
    int e = FindGCDEuclid(d, c);

    return e;

}
...
```

7. Declare another overloaded version of the **FindGCDEuclid** method. The method should accept four integer parameters called a, b, c, and d, and return an integer value. Use the other **FindGCDEuclid** method overloads to find the GCD of these parameters and return the result.

   Your code should resemble the following code example.

```
...
public static int FindGCDEuclid(int a, int b, int c, int d)
{
    int e = FindGCDEuclid(a, b, c);
    int f = FindGCDEuclid(e, d);
    return f;
}
...
```

8. Declare another overloaded version of the **FindGCDEuclid** method. The method should accept five integer parameters called a, b, c, d, and e, and return an integer value. Use the other **FindGCDEuclid** method overloads to find the GCD of these parameters and return the result.

   Your code should resemble the following code example.

```
...
public static int FindGCDEuclid(int a, int b, int c, int d, int e)
{
    int f = FindGCDEuclid(a, b, c, d);
    int g = FindGCDEuclid(f, e);
    return g;
}
...
```

   At the end of this task, the **GCDAlgorithms** class should resemble the following code example.

```
...
static class GCDAlgorithms
{
    ...
    // TODO Exercise 2, Task 2
    // Add overloaded methods for 3,4, and 5 integers
    public static int FindGCDEuclid(int a, int b, int c)
    {
        int d = FindGCDEuclid(a, b);
        int e = FindGCDEuclid(d, c);
        return e;
    }
    public static int FindGCDEuclid(int a, int b, int c, int d)
    {
        int e = FindGCDEuclid(a, b, c);
        int f = FindGCDEuclid(e, d);
        return f;
    }
    public static int FindGCDEuclid(int a, int b, int c, int d, int e)
    {
```

```
        int f = FindGCDEuclid(a, b, c, d);
        int g = FindGCDEuclid(f, e);
        return g;
    }
}
...
```

Task 3: Test the overloaded methods

1. Use the Task List window to navigate to the **TODO Exercise 2, Task 3** task.

   This task is located in the code for the WPF window that you can use to test your code:

   - In the Task List window, double-click **TODO Exercise 2, Task 3**.

2. Remove the **TODO Exercise 2, Task 3** comment, locate **the else if (sender == findGCD3)** block, and modify the statement that sets the **Content** property of the **resultEuclid** label to "N/A" as follows:

   a. Call the **FindGCDEuclid** overload that accepts three parameters and pass the variables firstNumber, secondNumber, and thirdNumber as arguments.

   b. Display the results in the **resultEuclid** label control. The result should be formatted as the following code example shows.

```
Euclid: result
```

   Your code should resemble the following code example.

```
...
else if (sender == findGCD3) // Euclid for three integers
{
    this.resultEuclid.Content =
        String.Format("Euclid: {0}",
        GCDAlgorithms.FindGCDEuclid(
            firstNumber,
            secondNumber,
            thirdNumber));
    this.resultStein.Content = "N/A";
}
...
```

3. Locate the **else if (sender == findGCD3)** block, the **else if (sender == findGCD4)** block, and the **else if (sender == findGCD5)** block, and modify

the statements that set the **Content** property of the **resultEuclid** label to "N/A". Call the appropriate **FindGCDEuclid** overload by using the firstNumber, secondNumber, thirdNumber, fourthNumber, and fifthNumber variables as arguments. Display the results in the **resultEuclid** label control.

Your code should resemble the following code example.

```
...
private void FindGCD_Click(object sender, RoutedEventArgs e)
{
    ...

    // TODO Exercise 2, Task 3
    // Call the overloaded methods for 3, 4 and 5 integers
    else if (sender == findGCD3) // Euclid for three integers
    {
        this.resultEuclid.Content =
            String.Format("Euclid: {0}",
            GCDAlgorithms.FindGCDEuclid(
            firstNumber, secondNumber, thirdNumber));
        this.resultStein.Content = "N/A";
    }
    else if (sender == findGCD4) // Euclid for four integers
    {
        this.resultEuclid.Content =
        String.Format("Euclid: {0}",
            GCDAlgorithms.FindGCDEuclid(
            firstNumber, secondNumber, thirdNumber, fourthNumber));
        this.resultStein.Content = "N/A";
    }
    else if (sender == findGCD5) // Euclid for five integers
    {
        this.resultEuclid.Content =
            String.Format("Euclid: {0}",
                GCDAlgorithms.FindGCDEuclid(firstNumber, secondNumber,
                thirdNumber, fourthNumber, fifthNumber));
        this.resultStein.Content = "N/A";
    }
}
...
```

4. Build the solution and correct any errors:
   - On the **Build** menu, click **Build Solution**. Correct any errors.
5. Run the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Start Debugging**.

6. In the GreatestCommonDivisor application, in the MainWindow window, type the values **7396 1978 1204 430 258** and then click **Find GCD (5 Integers)**.

   Verify that the result **86** is displayed.

7. Use the window to calculate the GCD for the values that are specified in the following table, and verify that the results that are displayed match those in the table.

First number	Second number	Third number	Fourth number	Fifth number	Result
2806	345	0	0	0	23
0	0	0	0	0	0
0	0	0	0	1	1
12	24	36	48	60	12
13	24	36	48	60	1
14	24	36	48	60	2
15	24	36	48	60	3
16	24	36	48	60	4
0	24	36	48	60	12

8. Close the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Stop Debugging**.

Task 4: Create unit tests for the overloaded methods

1. In Visual Studio, review the task list:
   a. If the task list is not already visible, on the **View** menu, click **Task List**.
   b. If the **Task List** is displaying **User Tasks**, in the drop-down list box click **Comments**.
2. Use the Task List window to navigate to the **TODO Exercise 2, Task 4** task:
   - In the Task List window, double-click **TODO Exercise 2, Task 4**.

3. Remove the **TODO Exercise 2, Task 4** comment and add a test method called **FindGCDEuclidTest1**.

   Your code should resemble the following code example.

```
...

[TestMethod()]
public void FindGCDEuclidTest1()
{

}

...
```

4. In the **FindGCDEuclidTest1** method, declare four variables called a, b, c, and expected, and assign them values **7396**, **1978**, **1204**, and **86** respectively.

   Your code should resemble the following code example.

```
[TestMethod()]
public void FindGCDEuclidTest1()
{
    int a = 7396;
    int b = 1978;
    int c = 1204;
    int expected = 86;
}
```

5. Declare a variable called actual, and assign it the result of a call to the **FindGCDEuclid** method call. Use the variables a, b, and c as arguments.

   Your code should resemble the following code example.

```
[TestMethod()]
public void FindGCDEuclidTest1()
{
    int a = 7396;
    int b = 1978;
    int c = 1204;
    int expected = 86;
    int actual = GCDAlgorithms.FindGCDEuclid(a, b, c);
}
```

6. Call the **AreEqual static** method of the **Assert** class, and pass the expected and actual variables as arguments.

   Your code should resemble the following code example.

```
[TestMethod()]

public void FindGCDEuclidTest1()
{
    int a = 7396;
    int b = 1978;
    int c = 1204;
    int expected = 86;
    int actual = GCDAlgorithms.FindGCDEuclid(a, b, c);
    Assert.AreEqual(expected, actual);
}
```

7. Repeat steps 4–6 to create two more test methods to test the other **FindGCDEuclid** method overloads. Create test methods called **FindGCDEuclidTest2** and **FindGCDEuclidTest3**. Use the values **7396**, **1978**, **1204**, and **430** for the **FindGCDEuclidTest2** method, and the values **7396**, **1978**, **1204**, **430**, and **258** for the **FindGCDEuclidTest3** method. The result should be **86** in both cases.

   Your code should resemble the following code example.

```
...
[TestClass()]
public class GCDAlgorithmsTest
{
    ...

    // Add unit tests for the new methods

    [TestMethod()]
    public void FindGCDEuclidTest1()
    {
        int a = 7396;
        int b = 1978;
        int c = 1204;
        int expected = 86;
        int actual = GCDAlgorithms.FindGCDEuclid(a, b, c);
        Assert.AreEqual(expected, actual);
    }

    [TestMethod()]
    public void FindGCDEuclidTest2()
    {
        int a = 7396;
        int b = 1978;
        int c = 1204;
        int d = 430;
        int expected = 86;
```

```
        int actual = GCDAlgorithms.FindGCDEuclid(a, b, c, d);
        Assert.AreEqual(expected, actual);
    }

    [TestMethod()]
    public void FindGCDEuclidTest3()
    {
        int a = 7396;
        int b = 1978;
        int c = 1204;
        int d = 430;
        int e = 258;
        int expected = 86;
        int actual = GCDAlgorithms.FindGCDEuclid(a, b, c, d, e);
        Assert.AreEqual(expected, actual);
    }
}
...
```

8. Open the Test View window and refresh the display if the unit test is not listed:
   a. On the **Test** menu, point to **Windows**, and then click **Test View**.
   b. In the Test View window, click the **Refresh** button.
9. Run the FindGCDEuclidTest, FindGCDEuclidTest1, FindGCDEuclidTest2, and FindGCDEuclidTest3 tests and verify that the tests ran successfully:
   a. In the Test View window, select the FindGCDEuclidTest, FindGCDEuclidTest1, FindGCDEuclidTest2, and FindGCDEuclidTest3 tests, right-click, and then click **Run Selection**.
   b. In the Test Results window, verify that the tests passed.

Exercise 3: Comparing the Efficiency of Two Algorithms

Task 1: Open the starter project

- Open the Stein solution in the E:\Labfiles\Lab 3\Ex3\Starter folder.

  This solution contains a completed copy of the code from Exercise 2:

  a. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
  b. In the **Open Project** dialog box, move to the **E:\Labfiles\Lab 3\Ex3\Starter** folder, click **Stein.sln**, and then click **Open**.

Task 2: Implement Stein's algorithm

1. Open the GCDAlgorithms.cs file:
   - In Solution Explorer, double-click **GCDAlgorithms.cs**.
2. At the end of the **GCDAlgorithms** class, remove the TODO comment and declare a **public static** method called **FindGCDStein**. The method should accept two integer parameters called u and v, and return an integer value.

   Your code should resemble the following code example.

```
...
static class GCDAlgorithms
{
    ...
    public static int FindGCDStein(int u, int v)
    {

    }
}
...
```

3. In the **FindGCDStein** method, add the code in the following code example, which calculates and returns the GCD of the values that are specified by the parameters u and v by using Stein's algorithm. You can either type this code manually, or use the Mod03Stein code snippet.

Note: For the purposes of this exercise, it is not necessary for you to understand this code. However, if you have time, you may like to compare this method to the algorithm that is described in the exercise scenario. Note that this code uses the left-shift (<<) and right-shift (>>) operators to perform fast multiplication and division by 2. If you left-shift an integer value by one place, the result is the same as multiplying the integer value by 2. Similarly, if you right-shift an integer value by one place, the result is the same as dividing the integer value by 2. In addition, the **|** operator performs a bitwise **OR** operation between two integer values. Consequently, if either *u* or *v* are zero, the expression **u | v** is a fast way of returning the value of whichever variable is non-zero, or zero if both are zero. Similarly, the **&** operator performs a bitwise **AND** operation, so the expression **u & 1** is a fast way to determine whether the value of *u* is odd or even.

```
static public int FindGCDStein(int u, int v)
{
    int k;

    // Step 1.
```

```
    // gcd(0, v) = v, because everything divides zero,
    // and v is the largest number that divides v.
    // Similarly, gcd(u, 0) = u. gcd(0, 0) is not typically
    // defined, but it is convenient to set gcd(0, 0) = 0.
    if (u == 0 || v == 0)
        return u | v;

    // Step 2.
    // If u and v are both even, then gcd(u, v) = 2·gcd(u/2, v/2),
    // because 2 is a common divisor.
    for (k = 0; ((u | v) & 1) == 0; ++k)
    {
        u >>= 1;
        v >>= 1;
    }

    // Step 3.
    // If u is even and v is odd, then gcd(u, v) = gcd(u/2, v),
    // because 2 is not a common divisor.
    // Similarly, if u is odd and v is even,
    // then gcd(u, v) = gcd(u, v/2).
    while ((u & 1) == 0)
        u >>= 1;

    // Step 4.
    // If u and v are both odd, and u ≥ v,
    // then gcd(u, v) = gcd((u − v)/2, v).
    // If both are odd and u < v, then gcd(u, v) = gcd((v − u)/2, u).
    // These are combinations of one step of the simple
    // Euclidean algorithm,
    // which uses subtraction at each step, and an application
    // of step 3 above.
    // The division by 2 results in an integer because the
    // difference of two odd numbers is even.
    do
    {
        while ((v & 1) == 0)  // Loop x
            v >>= 1;

        // Now u and v are both odd, so diff(u, v) is even.
        //   Let u = min(u, v), v = diff(u, v)/2.
        if (u < v)
        {
            v -= u;
        }
        else
        {
            int diff = u - v;
            u = v;
```

```
            v = diff;
        }
        v >>= 1;
        // Step 5.
        // Repeat steps 3-4 until u = v, or (one more step)
        // until u = 0.
        // In either case, the result is (2^k) * v, where k is
        // the number of common factors of 2 found in step 2.
    } while (v != 0);
    u <<= k;
    return u;
}
```

Task 3: Test the FindGCDStein method

1. Open the MainWindow.xaml.cs file:
   - In Solution Explorer, double-click **MainWindow.xaml.cs**.
2. In the **MainWindow** class, in the **FindGCD_Click** method, locate the **TODO Exercise 3, Task 2** comment. Remove this comment and replace the statement that sets the **Content** property of the **resultStein** label with code that calls the **FindGCDStein** method by using the variables firstNumber and secondNumber as arguments. Display the results in the **resultStein** label control. The result should be formatted as the following code example shows.

```
Stein: result
```

Your code should resemble the following code example.

```
if (sender == findGCD) // Euclid for two integers
{
    // Do the calculations

    this.resultEuclid.Content =
    String.Format("Euclid: {0}",
        GCDAlgorithms.FindGCDEuclid(firstNumber, secondNumber));
    // Call the method that implements Stein's algorithm and display
    // the results

    this.resultStein.Content =
    String.Format("Stein: {0}",
       GCDAlgorithms.FindGCDStein(firstNumber, secondNumber));
    ...
}
```

3. Build the solution and correct any errors:
   - On the **Build** menu, click **Build Solution**. Correct any errors.
4. Run the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Start Debugging**.
5. In the GreatestCommonDivisor application, in the MainWindow window, in the first two boxes, type the values **298467352** and **569484** and then click **Find GCD (2 Integers)**.

   Verify that the value **4** is displayed in both labels.
6. Close the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Stop Debugging**.
7. Open the GCDAlgorithmsTest.cs file:
   - In Solution Explorer, double-click **GCDAlgorithmsTest.cs**.
8. At the end of the **GCDAlgorithmsTest** class, locate the **TODO Exercise 3, Task 2** comment, remove the comment, and then add a test method called **FindGCDSteinTest**.

   Your code should resemble the following code example.

```
...
[TestMethod()]
public void FindGCDSteinTest()
{

}
...
```

9. In the **FindGCDSteinTest** method, declare three variables called u, v, and expected, and assign them values **298467352**, **569484**, and **4** respectively.

   Your code should resemble the following code example.

```
[TestMethod()]
public void FindGCDSteinTest()
{
    int u = 298467352;
    int v = 569484;
    int expected = 4;
}
```

10. Declare a variable called actual, and assign it the result of a call to the **FindGCDStein** method call. Use the variables u and v as arguments.

    Your code should resemble the following code example.

```
[TestMethod()]
public void FindGCDSteinTest()
{
    int u = 298467352;
    int v = 569484;
    int expected = 4;
    int actual = GCDAlgorithms.FindGCDStein(u, v);
}
```

11. Call the **static AreEqual** method of the **Assert** class, and pass the expected and actual variables as arguments.

    Your code should resemble the following code example.

```
[TestMethod()]
public void FindGCDSteinTest()
{
    int u = 298467352;
    int v = 569484;
    int expected = 4;
    int actual = GCDAlgorithms.FindGCDStein(u, v);
    Assert.AreEqual(expected, actual);
}
```

12. Open the Test View window and refresh the display if the unit test is not listed:

    a. On the **Test** menu, point to **Windows**, and then click **Test View**.

    b. In the Test View window, click the **Refresh** button.

13. Run the FindGCDSteinTest test, and verify that the test ran successfully:

    a. In the Test View window, right-click the **FindGCDSteinTest** test, and then click **Run Selection**.

    b. In the Test Results window, verify that the test passed.

Task 4: Add code to test the performance of the algorithms

1. Open the GCDAlgorithms.cs file:

    - In Solution Explorer, double-click **GCDAlgorithms.cs**.

2. In the **GCDAlgorithms** class, locate the **FindGCDEuclid** method that accepts two parameters, and modify the method signature to take an out parameter called time of type **long**.

   Your code should resemble the following code example.

```
...
static public int FindGCDEuclid(int a, int b, out long time)
{
}
...
```

3. At the start of the **FindGCDEuclid** method, add code to initialize the time parameter to zero, create a new **Stopwatch** object called **sw**, and start the stop watch.

   The **Stopwatch** class is useful for timing code. The **Start** method starts an internal timer running. You can subsequently use the **Stop** method to halt the timer, and establish how long the interval was between starting and stopping the timer by querying the **ElapsedMilliseconds** or **ElapsedTicks** properties.

   Your code should resemble the following code example.

```
...

static public int FindGCDEuclid(int a, int b, out long time)
{
    time = 0;
    Stopwatch sw = new Stopwatch();
    sw.Start();
    ...
}

...
```

4. At the end of the **FindGCDEuclid** method, before the **return** statement, add code to stop the **Stopwatch** object, and set the time parameter to the number of elapsed ticks of the **Stopwatch** object.

   Your code should resemble the following code example.

```
...

static public int FindGCDEuclid(int a, int b, out long time)
{
    time = 0;
    Stopwatch sw = new Stopwatch();
    sw.Start();
```

```
    ...
    sw.Stop();
    time = sw.ElapsedTicks;
    return a;
}
...
```

5. Comment out the other **FindGCDEuclid** method overloads.

6. Modify the **FindGCDStein** method to include the time output parameter, and add code to record the time each method takes to run. Note that the **FindGCDStein** method contains two **return** statements, and you should record the time before each one.

   Your code should resemble the following code example.

```
...
static public int FindGCDStein(int u, int v, out long time)
{
    time = 0;
    Stopwatch sw = new Stopwatch();
    sw.Start();
    int k;

    // Step 1.
    // gcd(0, v) = v, because everything divides zero, and
    // v is the largest number that divides v.
    // Similarly, gcd(u, 0) = u. gcd(0, 0) is not typically
    // defined, but it is convenient to set gcd(0, 0) = 0.
    if (u == 0 || v == 0)
    {
        sw.Stop();
        time = sw.ElapsedTicks;
        return u | v;
    }
    ...
    sw.Stop();
    time = sw.ElapsedTicks;
    return u;
}
...
```

7. Open the MainWindow.xaml.cs file:

   - In Solution Explorer, double-click **MainWindow.xaml.cs**.

8. In the **FindGCD_Click** method, modify each of the calls to the **FindGCDEuclid** method and the **FindGCDStein** method to use the updated method signatures, as follows:

   a. For calling the Euclid algorithm, create a long variable called timeEuclid.

   b. For calling the Stein algorithm, create a long variable called timeStein.

   c. Format the results displayed in the labels as the following code example shows.

```
[Euclid]
Euclid: result, Time (ticks): result

[Stein]
Stein: result, Time (ticks): result
```

   Your code should resemble the following code example.

```
...
if (sender == findGCD) // Euclid and Stein for two integers and graph
{
    long timeEuclid;
    long timeStein;

    // Do the calculations
    this.resultEuclid.Content =
        String.Format("Euclid: {0}, Time (ticks): {1}",
        GCDAlgorithms.FindGCDEuclid(firstNumber, secondNumber,
        out timeEuclid), timeEuclid);
    this.resultStein.Content = String.Format("Stein: {0}, Time
        (ticks): {1}", GCDAlgorithms.FindGCDStein(firstNumber,
        secondNumber, out timeStein), timeStein);
}
```

9. Comment out the code that calls the overloaded versions of the **FindGCDEuclid** method.

   Your code should resemble the following code example.

```
...
else if (sender == findGCD3) // Euclid for three integers
{
//    this.resultEuclid.Content = String.Format("Euclid: {0}",
//        GCDAlgorithms.FindGCDEuclid(firstNumber,
//        secondNumber, thirdNumber));
    this.resultStein.Content = "N/A";
}
else if (sender == findGCD4) // Euclid for four integers
```

```
{
//    this.resultEuclid.Content = String.Format("Euclid: {0}",
//        GCDAlgorithms.FindGCDEuclid(firstNumber,
//        secondNumber, thirdNumber, fourthNumber));
    this.resultStein.Content = "N/A";
}

else if (sender == findGCD5) // Euclid for five integers
{

//    this.resultEuclid.Content = String.Format("Euclid: {0}",
//        GCDAlgorithms.FindGCDEuclid(firstNumber,
//        secondNumber, thirdNumber, fourthNumber, fifthNumber));
    this.resultStein.Content = "N/A";
}

...
```

10. Open the GCDAlgorithmsTest.cs file:
    - In Solution Explorer, double-click **GCDAlgorithmsTest.cs**.
11. Modify the **FindGCDEuclidTest** and **FindGCDSteinTest** methods to use the new method signatures. Comment out the methods **FindGCDEuclidTest1**, **FindGCDEuclidTest2**, and **FindGCDEuclidTest3**.

    Your code should resemble the following code example.

```
[TestClass()]
public class GCDAlgorithmsTest
{
    ...

    [TestMethod()]
    public void FindGCDEuclidTest()
    {

        int a = 298467352;
        int b = 569484;
        long time;
        int expected = 4;
        int actual;
        actual = GCDAlgorithms.FindGCDEuclid(a, b, out time);
        Assert.AreEqual(expected, actual);
    }

    ...
    [TestMethod()]
    public void FindGCDSteinTest()
```

```
    {
        int u = 298467352;
        int v = 569484;
        long time;
        int expected = 4;
        int actual = GCDAlgorithms.FindGCDStein(u, v, out time);
        Assert.AreEqual(expected, actual);
    }

}
```

12. Build the solution and correct any errors:
    - On the **Build** menu, click **Build Solution**. Correct any errors.
13. Run the GreatestCommonDivisor application:
    - On the **Debug** menu, click Start **Debugging**.
14. In the GreatestCommonDivisor application, in the MainWindow window, in the first two boxes, type the values **298467352** and **569484** and then click **Find GCD (2 Integers)**. The result of **4** should be displayed. The time reported for Euclid's algorithm should be approximately three times more than that for Stein's algorithm.

Note: The bigger the difference between the two values, the more efficient Stein's algorithm becomes compared to Euclid's. If you have time, try experimenting with different values.

15. Close the GreatestCommonDivisor application:
    - On the **Debug** menu, click **Stop Debugging**.
16. Open the Test View window and refresh the display if the unit test is not listed:
    a. On the **Test** menu, point to **Windows**, and then click **Test View**.
    b. In the Test View window, click the **Refresh** button.
17. Run the **FindGCDEuclidTest** and **FindGCDSteinTest** methods and verify that the tests ran successfully:
    a. In the Test View window, select the **FindGCDEuclidTest** and **FindGCDSteinTest** tests, right-click, and then click **Run Selection**.
    b. In the Test Results window, verify that the tests passed.

Exercise 4: Displaying Results Graphically

Task 1: Open the starter project

- Open the Charting solution in the E:\Labfiles\Lab 3\Ex4\Starter folder.

  This solution contains a completed copy of the code from Exercise 3:

  a. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.

  b. In the **Open Project** dialog box, move to the **E:\Labfiles\Lab 3\Ex4\Starter** folder, click **Charting.sln**, and then click **Open**.

Task 2: Display the algorithm timings graphically

1. Open the MainWindow.xaml.cs file:

   - In Solution Explorer, double-click **MainWindow.xaml.cs**.

2. In the **FindGCD_Click** method, locate the **Call DrawGraph** comment, and add a call to the **DrawGraph** method, using the timeEuclid and timeStein variables as parameters.

   Your code should resemble the following code example.

```
...
if (sender == findGCD) // Euclid and Stein for two integers and graph
{
    ...
    DrawGraph(timeEuclid, timeStein);
}
...
```

3. Build the solution and correct any errors:

   - On the **Build** menu, click **Build Solution**. Correct any errors.

4. Run the GreatestCommonDivisor application:

   - On the **Debug** menu, click **Start Debugging**.

5. In the GreatestCommonDivisor application, in the MainWindow window, in the first two boxes, type the values **298467352** and **569484** and then click **Find GCD (2 Integers)**. The result of **4** should be displayed. The time reported for both algorithms should be represented by a simple bar graph in the window.

6. Close the GreatestCommonDivisor application:
    - On the **Debug** menu, click **Stop Debugging**.

Task 3: Modify the DrawGraph method

1. In the **MainWindow** class, locate the **DrawGraph** method and add the following three optional parameters:
    a. A parameter called orientation of type **Orientation** with a default value of **Orientation.Horizontal**.
    b. A parameter called colorEuclid of type **string** with a default value of **"Red"**.
    c. A parameter called colorStein of type **string** with a default value of **"Blue"**.

    Your code should resemble the following code example.

```
...

private void DrawGraph(long euclidTime, long steinTime,
    Orientation orientation = Orientation.Horizontal,
    string colorEuclid = "Red",
    string colorStein = "Blue")
{
    ...
}

...
```

2. In the **DrawGraph** method, locate the **Use optional orientation parameter** comment, and remove the existing declaration of the orientation variable.
3. Locate the **Use optional color parameters** comment, and modify the assignment of the bEuclid and bStein variables to use the optional parameters in the method signature. To do this, you will need to use the **BrushConverter** class and the **ConvertFromString** instance method as shown in the following code example.

```
...

private void DrawGraph(long euclidTime, long steinTime,
    Orientation orientation = Orientation.Horizontal,
    string colorEuclid = "Red",
    string colorStein = "Blue")
{
    ...
```

```
    BrushConverter bc = new BrushConverter();
    Brush bEuclid = (Brush)bc.ConvertFromString(colorEuclid);
    Brush bStein = (Brush)bc.ConvertFromString(colorStein);
    ...
}
...
```

4. Build the solution and correct any errors:
   - On the **Build** menu, click **Build Solution**. Correct any errors.
5. Run the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Start Debugging**.
6. In the GreatestCommonDivisor application, in the MainWindow window, in the first two boxes, type the values **298467352** and **569484** and then click **Find GCD (2 Integers)**. The graph should be displayed as before, except the **DrawGraph** method call is now using the default parameter values, and the graph is displayed as a pair of red and blue vertical bars.
7. Close the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Stop Debugging**.

Task 4: Modify the code that calls the DrawGraph method

1. Open the MainWindow.xaml.cs file:
   - In Solution Explorer, double-click **MainWindow.xaml.cs**.
2. In the **FindGCD_Click** method, locate the Modify the call to **Drawgraph to use the optional parameters** comment, and modify the **DrawGraph** method call to use the orientation, colorEuclid, and colorStein optional parameters as follows:
   a. *orientation*—set to the selected value of the **chartOrientation** list box.
   b. *colorEuclid*—set to the selected item of the **euclidColor** list box.
   c. *colorStein*—set to the selected item of the **steinColor** list box.

   These list boxes are already included in the user interface; they appear in the lower part of the window. The user can select the values in these list boxes to change the appearance of the graph that is displayed.

   Your code should resemble the following code example.

```
...
if (sender == findGCD) // Euclid and Stein for two integers and graph
{
    ...

    // Get the preferred colors and orientation
    string selectedEuclidColor =
        ((ListBoxItem)this.euclidColor.SelectedItem).
        Content.ToString();
    string selectedSteinColor =
        ((ListBoxItem)this.steinColor.SelectedItem).
        Content.ToString();

    Orientation orientation;

    if (this.chartOrientation.SelectedIndex == 0)
    {
        orientation = Orientation.Vertical;
    }
    else
    {
        orientation = Orientation.Horizontal;
    }

    DrawGraph(timeEuclid, timeStein, orientation: orientation,
        colorStein: selectedSteinColor,
        colorEuclid: selectedEuclidColor);
}
...
```

3. Build the solution and correct any errors:
   - On the **Build** menu, click **Build Solution**. Correct any errors.
4. Run the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Start Debugging**.
5. In the GreatestCommonDivisor application, in the MainWindow window, in the first two boxes, type the values **298467352** and **569484**
6. In the **Euclid** list box, select **Green**, in the **Stein** list box, select **Black**, in the **Orientation** box, select **Horizontal**, and then click **Find GCD (2 Integers)**. The graph should be displayed with the specified colors and direction.
7. Close the GreatestCommonDivisor application:
   - On the **Debug** menu, click **Stop Debugging**.

Exercise 5: Solving Simultaneous Equations (optional)

Task 1: Open the starter project

1. Open the SimultaneousEquations solution in the E:\Labfiles\Lab 3\Ex5 \Starter folder:

    a. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.

    b. In the **Open Project** dialog box, move to the **E:\Labfiles\Lab 3\Ex5 \Starter** folder, click **SimultaneousEquations.sln**, and then click **Open**.

2. Open the MainWindow.xaml file:

    - In Solution Explorer, expand the **GaussianElimination** project, and then double-click **MainWindow.xaml**.

    This is a different application from the one that the previous exercises have used. It is a WPF application that enables a user to enter the coefficients for four simultaneous equations that contain four variables (w, x, y, and z), and then uses Gaussian Elimination to find a solution for these equations. The results are displayed in the lower part of the screen.

Task 2: Create methods to copy arrays

1. Open the Gauss.cs file:

    - In Solution Explorer, double-click **Gauss.cs**.

    This file contains a class called **Gauss** that provides a method called **SolveGaussian**. This method takes two arrays as parameters:

    - A two-dimensional array of **double** values containing the coefficients for the variables w, x, y, and z specified by the user for each equation.
    - An array of **double** values containing the result of each equation specified by the user (the value to the right of the equal sign).

    The method returns an array of **double** values that will be populated with the values of w, x, y, and z that provide the solutions to these equations.

    You will implement the body of this method in this exercise.

2. In the **Gauss** class, locate the **TODO Exercise 5, Task 2** comment. Remove this comment and declare a **private static** method called **DeepCopy1D**. The method should accept and return a **double** array.

The **SolveGaussian** method will make a copy of the arrays passed in as parameters to avoid changing the original data that the user provided.

Your code should resemble the following code example.

```
...
private static double[] DeepCopy1D(double[] array)
{

}
...
```

3. In the **DeepCopy1D** method, add code to create a deep copy of the one-dimensional array that was passed into the method. Your code should perform the following tasks:

   a. Create and initialize an array with the same number of columns as the array that was passed in.

   b. Copy the values in the array that was passed as a parameter into the new array.

   c. Return the new array.

   Your code should resemble the following code example.

```
...
private static double[] DeepCopy1D(double[] array)
{
   // Get dimensions
    int columns = array.GetLength(0);

    // Initialize a new array
    double[] newArray = new double[columns];

    // Copy the values
    for (int i = 0; i < columns; i++)
    {
         newArray[i] = array[i];
    }
    return newArray;
}
...
```

4. In the **Gauss** class, declare another **private static** method called **DeepCopy2D**. The method should accept and return a two-dimensional **double** array.

   Your code should resemble the following code example.

```
...
private static double[,] DeepCopy2D(double[,] array)
{

}
...
```

5. In the **DeepCopy2D** method, add code to create a deep copy of the two-dimensional array that was passed into the method. Your code should do the following:

   a. Create and initialize an array with the same number of columns and rows as the array that was passed in.

   b. Copy the values in the array that was passed in as the parameter into the new array.

   c. Return the new array.

   Your code should resemble the following code example.

```
...

private static double[,] DeepCopy2D(double[,] array)
{
    // Get dimensions
    int columns = array.GetLength(0);
    int rows = array.GetLength(1);

    // Initialize a new array
    double[,] newArray = new double[columns, rows];

    // Copy the values
    for (int i = 0; i < columns; i++)
    {
        for (int j = 0; j < rows; j++)
        {
             newArray[i,j] = array[i,j];
        }
    }
    return newArray;
}

...
```

Task 3: Convert the equations to triangular form

1. In the **SolveGaussian** method, use the **DeepCopy1D** and **DeepCopy2D** methods to create deep copies of the **rhs** and **coefficients** arrays.

   Your code should resemble the following code example.

```
...
public static double[] SolveGaussian(double[,] coefficients,
    double[] rhs)
{
    // TODO Exercise 5, Task 3
    // Make deep copies of the coefficients and rhs arrays
    double[,] a = DeepCopy2D(coefficients);
    double[] b = DeepCopy1D(rhs);

    ...
}
...
```

2. Locate the **Convert the equation to triangular form** comment, and add code to convert the equations represented by the copies of the **coefficients** and **rhs** arrays into triangular form.

Note: The **Gauss** class defines a constant integer called **numberOfEquations** that specifies the number of coefficients that the application can resolve.

Your code should resemble the following code example.

```
...

// TODO Exercise 5, Task 3
// Convert the equations to triangular form

double x, sum;

for (int k = 0; k < numberOfEquations - 1; k++)
{

    try
    {

        for (int i = k + 1; i < numberOfEquations; i++)
        {

            x = a[i, k] / a[k, k];
            for (int j = k + 1; j < numberOfEquations; j++)
```

```
                a[i, j] = a[i, j] - a[k, j] * x;

            b[i] = b[i] - b[k] * x;
        }
    }

    catch (DivideByZeroException e)
    {
        Console.WriteLine(e.Message);
    }

}
...
```

Task 4: Perform back substitution

- In the **Gauss** class, in the **SolveGaussian** method, locate the **Perform the back substitution and return the result** comment, and then add code to perform back substitution. To do this, you will need to work back from the equation with one unknown and substituting the values calculated at each stage to solve the remaining equations.

  Your code should resemble the following code example.

```
...

// TODO Exercise 5, Task 4
// Perform the back substitution and return the result

b[numberOfEquations - 1] = b[numberOfEquations - 1] /
a[numberOfEquations - 1, numberOfEquations - 1];

for (int i = numberOfEquations - 2; i >= 0; i--)
{
    sum = b[i];
    for (int j = i + 1; j < numberOfEquations; j++)
        sum = sum - a[i, j] * b[j];
    b[i] = sum / a[i, i];
}

return b;

...
```

Task 5: Test the solution

1. Open the MainWindow.xaml.cs file:
   - In Solution Explorer, double-click **MainWindow.xaml.cs**.
2. In the **MainWindow** class, locate the **TODO Exercise 5, Step 5** comment, and add code to call the **SolveGaussion** method. Use the coefficients and rhs variables as parameters and set the **answers** array to the result.

   Your code should resemble the following code example.

```
...
private void CmdSolve_Click(object sender, RoutedEventArgs e)
{
    ...
    // TODO Exercise 5, Step 5
    // Invoke the solveGaussian method
    answers = Gauss.SolveGaussian(coefficients, rhs);
    ...
} ...
```

3. Run the GaussianElimination application:
   - On the **Debug** menu, click **Start Debugging**.
4. In the GaussianElimination application, in the MainWindow window, enter the following equations, and then click **Solve**.

Note: Enter a value of zero in the corresponding text if no value is specified for w, x, y, or z in the equations below.

$2w + x - y + z = 8$

$-3w - x + 2y + z = -11$

$-2w + x - 2y = -3$

$3w - x + 2y - 2z = -5$

Verify that the following results are displayed:

$w = 4$

$x = -17$

$y = -11$

$z = 6$

5. Experiment with other equations. Note that not all systems of equations have a solution. How does your code handle this situation?
6. Close the MainWindow window.
7. Close Visual Studio.