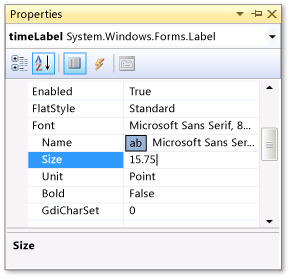
To create a project and add labels to your form

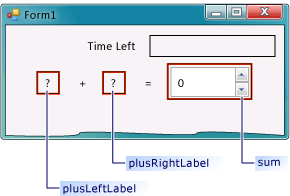
1. On the **File** menu, click **New Project**.
2. If you’re not using Visual Studio Express, you need to select a language first. From the **Installed Templates**list, select either **C#** or **Visual Basic**.
3. Click the **Windows Forms Application** icon, and then type **Math Quiz** as the name.
4. Set the form properties:
   1. Change the form's **Text** property to **Math Quiz**.
   2. Change the size to 500 pixels wide by 400 tall by using the **Size** property or dragging until you see the correct size in the lower-left corner of the integrated development environment (IDE).
   3. To prevent users from changing the form's size, change the **FormBorderStyle** property to **Fixed3D**and set the **MaximizeBox** property to **False**.
5. Drag a **Label** control from the Toolbox, and then set its properties:
   1. Change the **(Name)** property to **timeLabel**. This label appears as a box in the upper-right corner of your form that shows the number of seconds counting down for the quiz.
   2. Change the **AutoSize** property to **False** so you can size the box yourself.
   3. Change the **BorderStyle** property to **FixedSingle** to draw a line around the box.
   4. Set the **Size** property to **200, 30**.
   5. Drag the label to the upper-right corner of the form until the blue spacer lines appear.
   6. Clear the **Text** property by clicking **Text** in the **Properties** window and pressing the BACKSPACE key.
   7. Change the font size to **15.75**. Click the plus sign next to the **Font** property in the **Properties** window, which displays several properties including **Size**, as shown in the following picture.

Properties window showing font size



1. Next, drag another **Label** control from the Toolbox, and then set its properties:
   1. Change the font size to **15.75**.
   2. Set the **Text** property to **Time Left**.
   3. Drag it to line up just to the left of the **timeLabel** label.
2. Now, add the controls for the addition problem. Drag a **Label** control from the Toolbox, and then set its properties:
   1. Set the **Text** property to **?** (question mark).
   2. Set the **AutoSize** property to **False**.
   3. Set the **Size** property to **60, 50**.
   4. Change the font size to **18**.
   5. Change the **TextAlign** property to **MiddleCenter**.
   6. Change the **Location** property to **75, 75** to position it on the form.
   7. Change the **(Name)** property to **plusLeftLabel**.
3. Select the **plusLeftLabel** label and copy it. (Press Ctrl+C or from the **Edit** menu, select **Copy**.) Then, do the following:
   1. Paste it three times. (Press Ctrl+V or from the **Edit** menu, select **Paste**.)
   2. Arrange the three new labels so that the boxes are in a row to the right of the **plusLeftLabel** label, using the spacer lines to space them out and line them up.
   3. Change the second label's **Text** property to **+** (plus sign).
   4. Change the third label's **(Name)** property to **plusRightLabel**.
   5. Change the fourth label's **Text** property to **=** (equal sign).
4. Drag a **NumericUpDown** control from the Toolbox, and then do the following:
   1. Change the font size to **18**, and then make it narrower so that the width is **100**.
   2. Drag it until it lines up with the **Label** controls for the addition problem.
   3. Change the **(Name)** property to **sum**. (You learn more about the **NumericUpDown** control later.) The quiz now has a first row, as shown in the following picture.

First row of math quiz

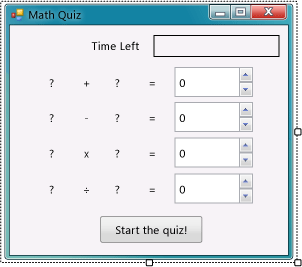


1. Select all five controls in your addition problem (the four **Label** controls and the **NumericUpDown** control) and copy them. Then, do the following:
   1. Paste the controls, which should add five new controls to your form.
   2. The controls should still be selected, so you can click one control and drag the controls into place so they're lined up underneath the addition controls. Use the spacer lines to give enough distance between the two rows.
   3. Change the second label's **Text** property to **–** (minus sign).
   4. Name the first question mark label **minusLeftLabel**.
   5. Name the second question mark label **minusRightLabel**.
   6. Name the **NumericUpDown** control **difference**.
2. Paste the five controls two more times, and then do the following:
   1. For the third row, name the first label **timesLeftLabel**, change the second label's **Text** property to **×**(multiplication sign), name the third label **timesRightLabel**, and name the **NumericUpDown** control**product**.
   2. For the fourth row, name the first label **dividedLeftLabel**, change the second label's **Text** property to**÷** (division sign), name the third label **dividedRightLabel**, and name the **NumericUpDown** control**quotient**.

|  |
| --- |
| **NoteNote** |
| You can copy the multiplication sign × and the division sign ÷ from this tutorial and paste them into the IDE. |

1. One more control is needed on your form: a button to start the quiz. Drag a **Button** control from the Toolbox, and then set its properties:
   1. Set the **(Name)** property to **startButton**.
   2. Set the **Text** property to **Start the quiz**.
   3. Set the font size to **14**.
   4. Set the **AutoSize** property to **True**, which causes the button to automatically resize to fit the text.
   5. Drag the button to the bottom of the form, and move it so that it's centered.
2. Finally, click the **startButton** control, and then do the following:
   1. Set the **TabIndex** property to **1**.
   2. Click the **NumericUpDown** sum control.
   3. Set the **TabIndex** property to **2**.
   4. Set the other **NumericUpDown** controls: Set the difference control's **TabIndex** property to **3**, the product control's **TabIndex** property to **4**, and the quotient control's **TabIndex** property to **5**. Now your form should look like the following picture.

Initial math quiz form



|  |
| --- |
| **NoteNote** |
| The purpose of the **TabIndex** property is to set the order of the controls when the user presses the TAB key. Open any dialog box (for example, from the **File** menu, select **Open**) and press the TAB key a few times. Watch how your cursor moves from control to control each time you press the TAB key. When that form was originally designed, a programmer decided the order. |

1. To see how the **TabIndex** property works, save and run your program, and then press the TAB key a few

To create a random addition problem

1. Create a **Random** object by using a **new** statement like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492147(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

public partial class Form1 : Form

{

// Create a Random object to generate random numbers.

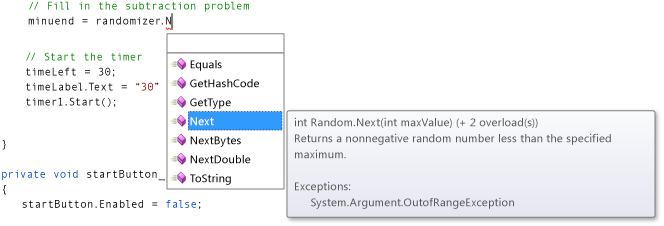
Random randomizer = new Random();

You have now added a **Random** object to your form and called it **randomizer**.

|  |
| --- |
| **NoteNote** |
| In the maze tutorial, you created two SoundPlayer components using the **new** statement. This does something similar. The only difference is that, unlike SoundPlayer, **Random** isn't a component, and it isn't a control, so it can't be called by those names. It's called an object. You've probably heard the word object before, and you learn more about what it means in the next few tutorials. For now, all you need to know is that when your program uses a **new** statement to create buttons, labels, panels, OpenFileDialogs, ColorDialogs, SoundPlayers, Randoms, and even forms, the item that gets created is called an object. Later tutorials show you much more about how these objects work. |

1. Now when you start your form, it creates a new **Random** object and gives it the name **randomizer**. Like with SoundPlayers, if you go into a method and start to type **randomizer**, and then dot (.), an **IntelliSense** window opens showing you all of the **Random** object's methods that you can call. You use the **Next()** method, as follows.

Next method



When you call **random.Next(50)**, you get a random number that's less than 50 (from 0 through 49).

1. Soon you will build a method to check the answers, so your program needs to remember what numbers it chose for the problems. Add an integer (known as an int in C# or Integer in Visual Basic) called **addend1** and an int (Integer) called **addend2** to the form (just like you added a **Random** object called **randomizer**), as follows.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492147(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

// Create a Random object to generate random numbers.

Random randomizer = new Random();

// These ints will store the numbers

// for the addition problem.

int addend1;

int addend2;

|  |
| --- |
| **NoteNote** |
| An int (Integer) is used to store a positive or negative number value. It can hold any number from -2147483648 through 2147483647. It can only store whole numbers, and not decimals. |

1. Next, add a method called **StartTheQuiz()** that uses the **Random** object **Next()** method to choose two numbers and place the numbers in the labels. It will eventually fill in all of the problems and then start the timer, so add a comment. It should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492147(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

/// <summary>

/// Start the quiz by filling in all of the problems

/// and starting the timer.

/// </summary>

public void StartTheQuiz()

{

// Fill in the addition problem.

addend1 = randomizer.Next(51);

addend2 = randomizer.Next(51);

plusLeftLabel.Text = addend1.ToString();

plusRightLabel.Text = addend2.ToString();

sum.Value = 0;

}

|  |
| --- |
| **NoteNote** |
| Note that you called **randomizer.Next(51)**. The reason 51 and not 50 is used is so the two numbers add up to an answer that's from 0 through 100. If you pass 50 to the **Next()** method, it chooses a number from 0 through 49, so the highest possible answer is 98, and not 100. After the first two statements in the method execute, each of the two ints (Integers), addend1 and addend2, hold a random number from 0 through 50. |

Take a closer look at these statements.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492147(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

plusLeftLabel.Text = addend1.ToString();

plusRightLabel.Text = addend2.ToString();

The statements set the **Text** properties of the two plus labels, **plusLeftLabel** and **plusRightLabel**, so that the labels display the two random numbers. You need to use the int's (Integer's) **ToString()** method to convert it to text (in programming, *string* means text), because **Label** controls only display text, and not numbers.

1. You want the **Start** button to start the quiz, so go to Windows Forms Designer and double-click the button to add a **Click** event handler. Then add the following two statements.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492147(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

private void startButton\_Click(object sender, EventArgs e)

{

StartTheQuiz();

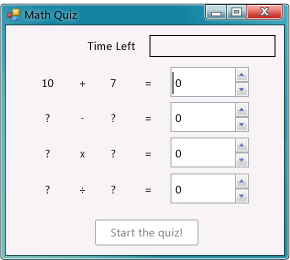
startButton.Enabled = false;

}

You know what the first statement does: It calls the new **StartTheQuiz()** method. The second statement sets the **startButton** control **Enabled** property to **False**. That disables the button, so the user can't click it. That way, the user can only click the **Start** button once. After that, the button appears dimmed and is unavailable, and the user must finish the quiz before the time runs out (or close the program).

1. Now save and run your program. Click the **Start** button. A random addition problem should appear, as shown in the following picture.

Random addition problem



To continue or review

To add a countdown timer

1. Add an int (Integer) called **timeLeft**, just like you did previously. Your code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492144(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

public partial class Form1 : Form

{

// Create a Random object to generate random numbers.

Random randomizer = new Random();

// These ints will store the numbers

// for the addition problem.

int addend1;

int addend2;

// This int will keep track of the time left.

int timeLeft;

1. Now you need something that actually does the counting, such as a timer. Go to Windows Forms Designer and drag a **Timer** control from the Toolbox (from the **Components** category) to your form. It will appear in the gray area at the bottom of Windows Forms Designer.
2. Click the **timer1** icon you just added, and set the **Interval** property to **1000**. This causes the Tick event to fire every second. Then double-click the icon to add the Tick event handler. The IDE switches to the code editor and jumps to the new event handler method. Add the following statements.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492144(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

private void timer1\_Tick(object sender, EventArgs e)

{

if (timeLeft > 0)

{

// Display the new time left

// by updating the Time Left label.

timeLeft = timeLeft - 1;

timeLabel.Text = timeLeft + " seconds";

}

else

{

// If the user ran out of time, stop the timer, show

// a MessageBox, and fill in the answers.

timer1.Stop();

timeLabel.Text = "Time's up!";

MessageBox.Show("You didn't finish in time.", "Sorry");

sum.Value = addend1 + addend2;

startButton.Enabled = true;

}

}

Based on what you added, each second the timer checks whether time has run out by checking whether the**timeLeft** int (Integer) is greater than 0. If it is, there's time left. First the timer subtracts 1 from timeLeft, and then it updates the **timeLabel** control **Text** property to show the user how many seconds are left.

|  |
| --- |
| **NoteNote** |
| The **timer1\_Tick** method is called after each tick interval. In this case, with an interval of 1000 milliseconds, the method is called once per second. You can set a breakpoint in this method to verify that behavior. Remember that time continues to pass during the period that you are stopped in the debugger, so it might appear as if the method is called again immediately. |

If there's no time left, the timer stops and changes the **timeLabel** control text so it shows **Time's up!** A message box appears telling the user that the quiz is over. The answer is revealed—in this case, by adding addend1 and addend2. The **startButton** control **Enabled** property is set to true, to make the button available again. That way, the user can start the quiz again.

You just added an **if else** statement, which is how you tell programs to make decisions. An **if else** statement looks like the following.

VB

If (something your program will check) Then

' statements that will get executed

' if the thing that the program checked is true

Else

' statements that will get executed

' if the thing that the program checked is NOT true

End If

C#

if (something your program will check)

{

// statements that will get executed

// if the thing that the program checked is true

}

else

{

// statements that will get executed

// if the thing that the program checked is NOT true

}

Take a closer look at the statement you added in the else block to show the answer to the addition problem.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492144(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

sum.Value = addend1 + addend2;

As you probably know, addend1 + addend2 adds the two values together. The first part (sum.Value) uses the**Value** property of the **NumericUpDown** control to display the correct answer. The **Value** property is also used later, when you want to check the answers for the quiz.

A **NumericUpDown** control makes it easy for users to enter numbers, which is why you use the control for the answers to the math problems. Because all of the answers are numbers from 0 through 100, you leave the default **Minimum** and **Maximum** properties set to 0 and 100. This causes the control to only allow a user to enter a number from 0 through 100. Because the answers can only be whole numbers, you leave the**DecimalPlaces** property set to 0, which means that the user can't enter decimals. (If you wanted to allow the user to enter 3.141 but not 3.1415, you could set the **DecimalPlaces** property to 3.)

1. Add three lines to the end of the **StartTheQuiz()** method, so the code looks like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492144(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

/// <summary>

/// Start the quiz by filling in all of the problems

/// and starting the timer.

/// </summary>

public void StartTheQuiz()

{

// Fill in the addition problem.

addend1 = randomizer.Next(51);

addend2 = randomizer.Next(51);

plusLeftLabel.Text = addend1.ToString();

plusRightLabel.Text = addend2.ToString();

sum.Value = 0;

// Start the timer.

timeLeft = 30;

timeLabel.Text = "30 seconds";

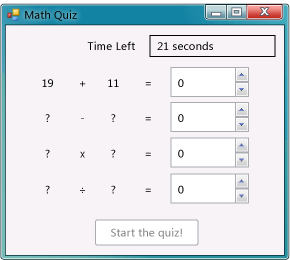
timer1.Start();

}

Now, when your quiz starts, it sets the **timeLeft** int (Integer) to 30, and changes the **timeLabel** control **Text**property to 30 seconds. Then it calls the **Timer** control's **Start()** method to start the countdown. (It doesn't check the answer yet—that's coming next.)

1. Save and run your program. When you click the **Start** button, the timer should start counting down. When time runs out, the quiz ends, and the answer appears. The following picture shows the quiz in progress.

Math quiz in progress



To add the CheckTheAnswer() method

1. Add the **CheckTheAnswer()** method, which adds addend1 and addend2 and verifies whether the sum is equal to the value in the sum **NumericUpDown** control. If the sum is equal, the method returns true; if not, it returns false. Your code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492165(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

/// <summary>

/// Check the answer to see if the user got everything right.

/// </summary>

/// <returns>True if the answer's correct, false otherwise.</returns>

private bool CheckTheAnswer()

{

if (addend1 + addend2 == sum.Value)

return true;

else

return false;

}

Your program needs to call this method to verify whether the user answered correctly. You do this by adding to your **if else** statement. The statement looks like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492165(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

if (CheckTheAnswer())

{

// statements that will get executed

// if the answer is correct

}

else if (timeLeft > 0)

{

// statements that will get executed

// if there's still time left on the timer

}

else

{

// statements that will get executed if the timer ran out

}

1. Next, you modify the timer's Tick event handler to check the answer. The new event handler with answer checking should include the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492165(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

private void timer1\_Tick(object sender, EventArgs e)

{

if (CheckTheAnswer())

{

// If the user got the answer right, stop the timer

// and show a MessageBox.

timer1.Stop();

MessageBox.Show("You got all the answers right!",

"Congratulations");

startButton.Enabled = true;

}

else if (timeLeft > 0)

{

// Decrease the time left by one second and display

// the new time left by updating the Time Left label.

timeLeft--;

timeLabel.Text = timeLeft + " seconds";

}

else

{

// If the user ran out of time, stop the timer, show

// a MessageBox, and fill in the answers.

timer1.Stop();

timeLabel.Text = "Time's up!";

MessageBox.Show("You didn't finish in time.", "Sorry");

sum.Value = addend1 + addend2;

startButton.Enabled = true;

}

}

Now if the timer's event handler finds that the user answered correctly, the event handler stops the timer, shows a congratulations message, and makes the **Start** button available again.

1. Save and run your program. Start the game, and type the correct answer to the addition problem.

|  |
| --- |
| **NoteNote** |
| When you type your answer, you may notice something odd about the **NumericUpDown** control. If you start typing without selecting the entire answer, the zero remains, and you must delete it manually. You will correct this later in this tutorial. |

1. When you type the correct answer, the message box should open, the **Start** button should be available, and the timer should stop. Click the **Start** button again and be sure this happens.

To see the NumericUpDown control behavior

1. Run your program and start the game. The sum **NumericUpDown** control should have a flashing cursor in it next to **0** (zero).
2. Type **3**, and **30** appears. Type **5**, and **350** appears, but after a second, it changes to **100**.

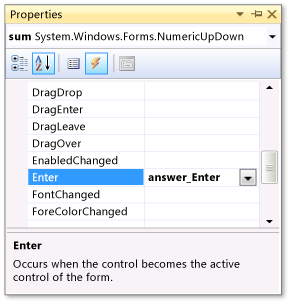
Before you fix this problem, think about what's happening. Consider why the **0** didn't disappear when you typed **3.** Consider why **350** changed to **100**, and why there was a delay before it changed.

|  |
| --- |
| **NoteNote** |
| Although it may seem like odd behavior, there is an explanation. When you click the **Start** button, the button's **Enabled** property is set to **False**, and the button appears dimmed and is unavailable. Your program looks for the control with the next lowest TabIndex value—the sum **NumericUpDown** control—and changes focus to that control. When you use the TAB key to go to a **NumericUpDown** control, it automatically positions the cursor at the start of the control, which is what causes the numbers that you type to be entered from the left and not the right. When you enter a number above the **MaximumValue**property, which is set to 100, it replaces the number entered with the maximum value. |

To add an Enter event handler for the NumericUpDown control

1. To prevent this odd behavior and make your program easier to use, add an event handler for each**NumericUpDown** control Enter event. Use the **Events** page in the **Properties** dialog box to add an Enter event handler for the sum **NumericUpDown** control called **answer\_Enter**.

Properties dialog box



The code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492138(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

private void answer\_Enter(object sender, EventArgs e)

{

// Select the whole answer in the NumericUpDown control.

NumericUpDown answerBox = sender as NumericUpDown;

if (answerBox != null)

{

int lengthOfAnswer = answerBox.Value.ToString().Length;

answerBox.Select(0, lengthOfAnswer);

}

}

Although it may look complex initially, it's easier to understand if you look at it step by step. First, look at the top of the method: object sender in C# or sender As System.Object in Visual Basic. This means that inside your method, any time you use sender, it refers to the **NumericUpDown** control whose Enter event is firing. So, in the first line of the method, you specify that it isn't just an object, but specifically a **NumericUpDown**control. (Every **NumericUpDown** control is an object, but not every object is a **NumericUpDown** control.) The next line verifies whether answerBox was successfully converted (cast) from an object to a **NumericUpDown**control. If unsuccessful, it would have a value of null (C#) or Nothing (Visual Basic). The third line finds the length of the answer that's currently displayed in the **NumericUpDown** control. The fourth line tells the**NumericUpDown** control to select the answer. Now when the user navigates into the control, it fires this event, which causes it to select the answer. As soon as the user starts typing, the previous answer is cleared and replaced with the new answer.

1. After this event handler is in place, go to Windows Forms Designer and select the difference**NumericUpDown** control. Go to the **Events** page in the **Properties** dialog box, scroll down to the Enter event, and select the event handler that you just added.
2. Then do the same for the product and quotient **NumericUpDown** controls.
3. Save and run your program. The odd behavior should no longer occur.

To add a subtraction problem

1. First, you need a place to store the values, so add two ints (Integers) for the subtraction problem to your form. The new code appears between the addition integers and the timer integer. The code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492142(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

public partial class Form1 : Form

{

// Create a Random object to generate random numbers.

Random randomizer = new Random();

// These ints will store the numbers

// for the addition problem.

int addend1;

int addend2;

// These ints will store the numbers

// for the subtraction problem.

int minuend;

int subtrahend;

// This int will keep track of the time left.

int timeLeft;

|  |
| --- |
| **NoteNote** |
| The names of the new ints—**minuend** and **subtrahend**—aren't programming terms. They are the traditional names in arithmetic for the number that's being subtracted (the subtrahend) and the number being subtracted from (the minuend). The difference is the minuend minus the subtrahend. You could use other names, because your program doesn't require specific names for ints, controls, components, or methods. There are some rules (for example, names can't begin with digits), but in general, you could use names such as x1, x2, x3, x4, and so on. But it would be difficult to read the code, and nearly impossible to track problems. You will use the traditional names for multiplication (multiplicand × multiplier = product) and division (dividend ÷ divisor = quotient) later in this tutorial. |

1. Next, modify the **StartTheQuiz()** method to fill in a random subtraction problem. The new code follows the "Fill in the subtraction problem" comment. The code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492142(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

/// <summary>

/// Start the quiz by filling in all of the problems

/// and starting the timer.

/// </summary>

public void StartTheQuiz()

{

// Fill in the addition problem.

addend1 = randomizer.Next(51);

addend2 = randomizer.Next(51);

plusLeftLabel.Text = addend1.ToString();

plusRightLabel.Text = addend2.ToString();

sum.Value = 0;

// Fill in the subtraction problem.

minuend = randomizer.Next(1, 101);

subtrahend = randomizer.Next(1, minuend);

minusLeftLabel.Text = minuend.ToString();

minusRightLabel.Text = subtrahend.ToString();

difference.Value = 0;

// Start the timer.

timeLeft = 30;

timeLabel.Text = "30 seconds";

timer1.Start();

}

This code uses the **Random** class **Next()** method a little differently. When you give it two values, it picks a random number that's greater than or equal to the first one and less than the second one. The following line chooses a random number from 1 through 100, and stores it in minuend.

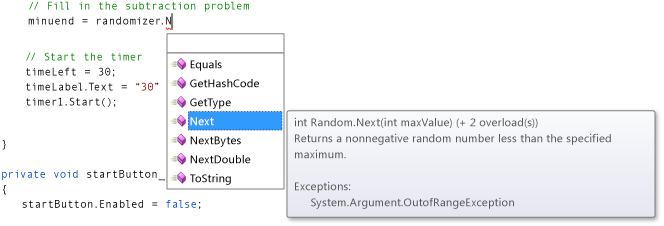
C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492142(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

minuend = randomizer.Next(1, 101);

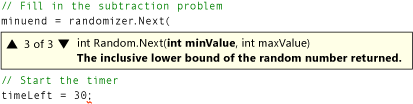
The **Random** class **Next()** method can be called in several ways. When you can call a method in more than one way, that's called an *overloaded method*, and you can use IntelliSense to explore this. Take another look at the **IntelliSense** window tooltip for the **Next()** method.

Intellisense window tooltip



Notice how the tooltip shows **(+ 2 overload(s))**. This means that there are two other ways that you can call the **Next()** method. When you type the new code for the **StartTheQuiz()** method, you can see more information. As soon as you type randomizer.Next(, the IntelliSense window opens. Press the UP ARROW and DOWN ARROW keys to cycle through the overloads, as shown in the following picture.

Intellisense window overloads



The one in the preceding picture is the one you want, because it lets you specify a minimum and maximum value.

1. Modify the **CheckTheAnswer()** method to check for the correct subtraction answer. The code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492142(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

/// <summary>

/// Check the answer to see if the user got everything right.

/// </summary>

/// <returns>True if the answer's correct, false otherwise.</returns>

private bool CheckTheAnswer()

{

if ((addend1 + addend2 == sum.Value)

&& (minuend - subtrahend == difference.Value))

return true;

else

return false;

}

The **&&** is the Visual C# **logical and** operator. In Visual Basic, the equivalent operator is **AndAlso**. It's the same as saying, "If addend1 plus addend2 is equal to the value of the sum NumericUpDown, and if minuend minus subtrahend is equal to the value of the difference NumericUpDown." The **CheckTheAnswer()** method only returns true if the addition problem is correct and the subtraction problem is correct.

1. Change the last part of the timer's Tick event handler so it fills in the correct answer when time runs out. The code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492142(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

else

{

// If the user ran out of time, stop the timer, show

// a MessageBox, and fill in the answers.

timer1.Stop();

timeLabel.Text = "Time's up!";

MessageBox.Show("You didn't finish in time.", "Sorry");

sum.Value = addend1 + addend2;

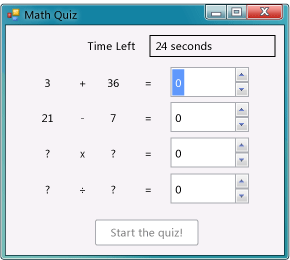
difference.Value = minuend - subtrahend;

startButton.Enabled = true;

}

1. Save and run your code. Your program should now have a subtraction problem, as shown in the following picture.

Math quiz with subtraction problem



To add multiplication and division problems

1. Add four ints (Integers) to the form. The code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492175(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

public partial class Form1 : Form

{

// Create a Random object to generate random numbers.

Random randomizer = new Random();

// These ints will store the numbers for the addition problem.

int addend1;

int addend2;

// These ints will store the numbers for the subtraction problem.

int minuend;

int subtrahend;

// These ints will store the numbers for the multiplication problem.

int multiplicand;

int multiplier;

// These ints will store the numbers for the division problem.

int dividend;

int divisor;

// This int will keep track of the time left.

int timeLeft;

1. Like before, modify the **StartTheQuiz()** method to fill in random multiplication and division problems. The code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492175(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

/// <summary>

/// Start the quiz by filling in all of the problems

/// and starting the timer.

/// </summary>

public void StartTheQuiz()

{

// Fill in the addition problem.

addend1 = randomizer.Next(51);

addend2 = randomizer.Next(51);

plusLeftLabel.Text = addend1.ToString();

plusRightLabel.Text = addend2.ToString();

sum.Value = 0;

// Fill in the subtraction problem.

minuend = randomizer.Next(1, 101);

subtrahend = randomizer.Next(1, minuend);

minusLeftLabel.Text = minuend.ToString();

minusRightLabel.Text = subtrahend.ToString();

difference.Value = 0;

// Fill in the multiplication problem.

multiplicand = randomizer.Next(2, 11);

multiplier = randomizer.Next(2, 11);

timesLeftLabel.Text = multiplicand.ToString();

timesRightLabel.Text = multiplier.ToString();

product.Value = 0;

// Fill in the division problem.

divisor = randomizer.Next(2, 11);

int temporaryQuotient = randomizer.Next(2, 11);

dividend = divisor \* temporaryQuotient;

dividedLeftLabel.Text = dividend.ToString();

dividedRightLabel.Text = divisor.ToString();

quotient.Value = 0;

// Start the timer.

timeLeft = 30;

timeLabel.Text = "30 seconds";

timer1.Start();

}

1. Modify the **CheckTheAnswer()** method so that the method also checks the multiplication and division problems. The code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492175(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

/// <summary>

/// Check the answer to see if the user got everything right.

/// </summary>

/// <returns>True if the answer's correct, false otherwise.</returns>

private bool CheckTheAnswer()

{

if ((addend1 + addend2 == sum.Value)

&& (minuend - subtrahend == difference.Value)

&& (multiplicand \* multiplier == product.Value)

&& (dividend / divisor == quotient.Value))

return true;

else

return false;

}

|  |
| --- |
| **NoteNote** |
| Because there's no simple way to enter the multiplication sign (×) and the division sign (÷) using the keyboard, Visual C# and Visual Basic use an asterisk (\*) for multiplication and a slash mark (/) for division. |

1. Change the last part of the timer's Tick event handler so it fills in the correct answer when time runs out. The code should look like the following.

C#

[**VB**](http://msdn.microsoft.com/en-us/library/dd492175(v=vs.100).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

else

{

// If the user ran out of time, stop the timer, show

// a MessageBox, and fill in the answers.

timer1.Stop();

timeLabel.Text = "Time's up!";

MessageBox.Show("You didn't finish in time.", "Sorry");

sum.Value = addend1 + addend2;

difference.Value = minuend - subtrahend;

product.Value = multiplicand \* multiplier;

quotient.Value = dividend / divisor;

startButton.Enabled = true;

}

1. Save and run your program. Now, four problems must be answered to complete the quiz, as shown in the following picture.

Math quiz with four problems

