Looping the Loop

Loops are an important part of any programming language, and C# is no different. A loop is a way to execute a piece of code repeatedly. The idea is that you go round and round until an end condition is met. Only then is the loop broken. As an example, suppose you want to add up the numbers one to ten. You could do it like this:

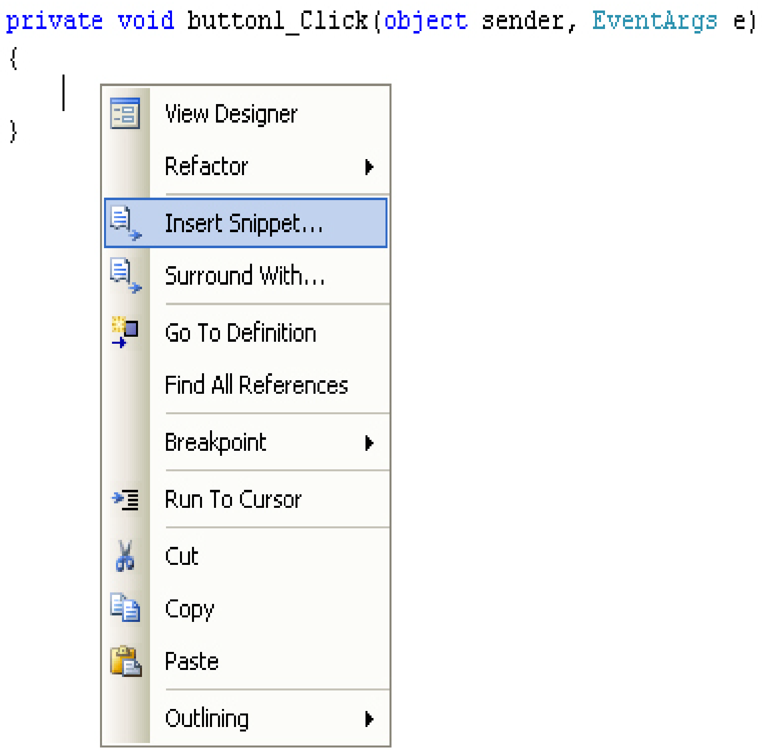
```
int answer;
answer = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;
```

Moreover, this would be OK if you only had 10 numbers. However, suppose you had a thousand number, or ten thousand. You are certainly not going to want to type them all out! Instead, you use a loop to repeatedly add the numbers.

For Loops

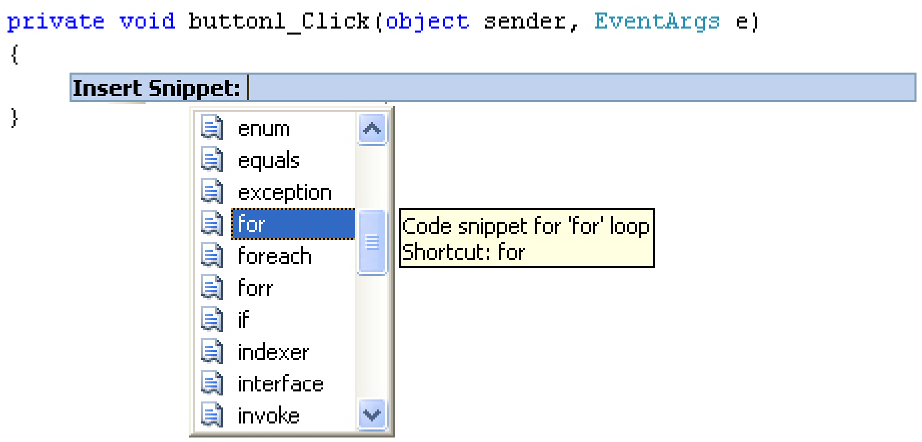
The first type of loop we will explore is called a FOR loop. Other types are do loops and while loops, which you will meet shortly. However, the FOR loop is the most common type of loop you will need. Let us use one to add up the numbers 1 to 100.

Start a new project by clicking **File** > **New Project** from the menu bars at the top of Visual Studio. Now add a button to the new form. Double click the button to get at the code. To quickly add a code stub for a loop, right click anywhere between the curly brackets of the button code. From the menu that appears, click on **Insert Snippet**:



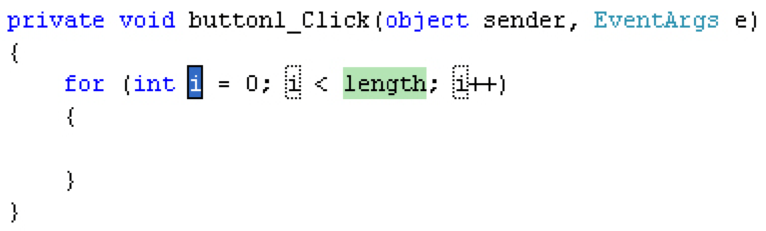
```
private void button1_Click(object sender, EventArgs e)
{

}
```

When you click on Insert Snippet, you will see a list of items (click the C# entry in version 2012, first):



```
private void button1_Click(object sender, EventArgs e)
{
    Insert Snippet:
}
```

Scroll down and double click on **for**. Some code is added for you:



```
private void button1_Click(object sender, EventArgs e)
{
    for (int i = 0; i < length; i++)
    {

    }
}
```

It all looks a bit complicated, so we will go through it. Here is the for loop without anything between the round brackets:

```
for ( )
{
}
```

Therefore, you start with the word for, followed by a pair of round brackets. What you are doing between the round brackets is telling C# how many times you want to go round the loop. After the round brackets, you type a pair of curly brackets. The code that you want to execute repeatedly goes between the curly brackets.

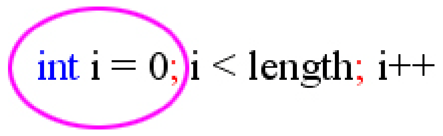
The default round-bracket code that C# inserts for you is this:

```
int i = 0; i < length; i++
```

There's three parts to the round-bracket code:

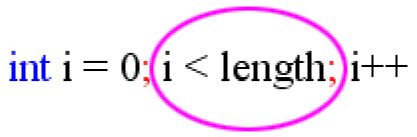
- ✓ Which number do you want your loop to start at?
- ✓ How many times do you want to go round and round?
- ✓ How do you want to update each time round the loop?

Note that each of the three parts separated by a semi-colon. Here is the first part:



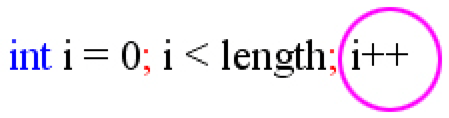
```
int i = 0; i < length; i++
```

And here is the second part:



```
int i = 0; i < length; i++
```

And here is the third part:



```
int i = 0; i < length; i++
```

Number 1 on the list above (Which number do you want to start at?) is this:

```
int i = 0;
```

What the default code is doing is setting up an integer variable called **i** (a popular name for loop variables.) It is then assigning a value of 0 to the **i** variable. It will use the value in **i** as the starting value of the loop. You can set up your starting variable outside the code, if you prefer. Like this:

```
int i
for (i = 0; i < length; i++)
{
}
```

So the variable called **i** is now set up outside the loop. We then just need to assign a value to the variable for the first part of the loop. Number 2 on the list above (How many times do you want to go round and round?) was this:

```
i < length;
```

This, if you remember your Conditional Logic from the previous section, says, "**i** is less than length". But length is not a keyword. Therefore, you need to either set up a variable called length, or replace the word length with a number. So either this:

```
for (int i = 0; i < 101; i++)
```

Or this:

```
int length = 101;
for (int i = 0; i < length; i++)
{
}
```

In the first example, we have just typed the i < 101. In the second example, we have set up a variable called length, and stored 101 in it. We are then just comparing one variable to another, and checking that **i** is less than **length**:

i < length;

If **i** is less than **length**, then the end condition has NOT been met and C# will keep looping. In other words, "Keep going round and round while **i** is less than **length**". However, you do not need to call the variable **length**. It is just a variable name, so you can come up with your own. For example:

```
int endNumber = 101;

for (int i = 0; i < endNumber; i++)
{
}
```

Here, we have called the variable **endNumber** instead of **length**. The second part now says, "Keep looping while **i** is less than **endNumber**". Number 3 on the list above (How do you want to update each time round the loop?) was this:

i++

This final part of a **for** loop is called the Update Expression. For the first two parts, you set a start value, and an end value for the loop. However, C# does not know how to get from one number to the other. You have to tell it how to get there. By typing **i++**, you are adding 1 to the value inside of **i** each time round the loop (called incrementing the variable). This:

variable_name++

Is a shorthand way of saying this:

variable_name = variable_name + 1

All you are doing is adding 1 to whatever is already inside of the variable name. Since you are in a loop, C# will keep adding 1 to the value of **i** each time round the loop. It only stops adding 1 to **i** when the end condition has been reached (**i** is no longer less than **length**). Therefore, to recap, you need a start value for the loop, how many times you want to go round and round, and how to get from one number to the other.

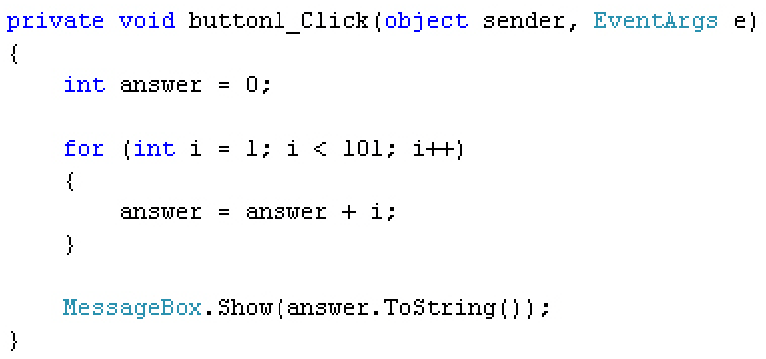
Therefore, your three parts are these:

for (**Start_Value**; **End_Value**; **Update_Expression**)

OK, time to put the theory into practice. Type the following for your button code:

```
int answer = 0;

for (int i = 1; i < 101; i++)
{
answer = answer + i;
}

MessageBox.Show(answer.ToString());
```

Your coding window should then look like this, when you're finished:



```
private void button1_Click(object sender, EventArgs e)
{
    int answer = 0;

    for (int i = 1; i < 101; i++)
    {
        answer = answer + i;
    }

    MessageBox.Show(answer.ToString());
}
```

The actual code for the loop, the code that goes inside of the curly brackets, is this:

answer = answer + i;

This is probably the trickiest part of loops – knowing what to put for your code! Just remember what you are trying to do: force C# to execute a piece of code a set number of times. We want to add up the numbers (1 to 100), and use a variable called **answer** to store the answer to the addition. Because the value in **i** is increasing by one each time round the loop, we can use this value in the addition. Here are the values the first time round the loop:

answer	=	**answer**	+	**i**	;
0	=	0	+	1	

The second time round the loop, the figures are these:

answer	=	**answer**	+	**i**	;
1	=	1	+	2	

The third time round the loop:

answer	=	**answer**	+	**i**	;
3	=	3	+	3	

And the fourth:

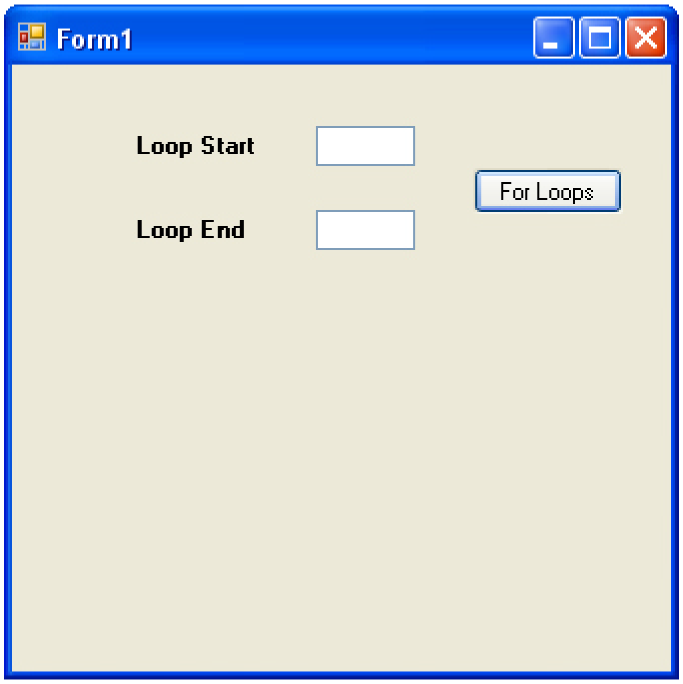
answer	=	**answer**	+	**i**	;
6	=	6	+	4	

Notice how the value of **i** increases by one each time round the loop. If you first do the addition after the equals sign, the above will make more sense! (As an exercise, what is the value of **answer** the fifth time round the loop?).

Run your program, and click the button. The message box should display an answer of 5050.

Loop Start Values and Loop End Values

In the code above, we typed the start value and end value for the loop. You can also get these from text boxes. Add two text boxes to your form. Add a couple of labels, as well. For the first label, type **Loop Start**. For the second label, type **Loop End**. Your form will then look something like this:



What we will do is to get the start value and end value from the text boxes. We will then use these in our **for** loop. So double click your button to get at the code. Set up two variables to hold the numbers from the text boxes:

```
int loopStart;
int loopEnd;
```

Now store the numbers from the text boxes into the two new variables:

```
loopStart = int.Parse(textBox1.Text);
loopEnd = int.Parse(textBox2.Text);
```

Now that we have the numbers from the text boxes, we can use them in the for loop. Change you for loop to this:

```
for (int i = loopStart; i < loopEnd; i++)
{
answer = answer + i;
}
```

The only thing you are changing here is the part between the round brackets. The first part has now changed from this:

```
int i = 1
```

To this:

```
int i = loopStart
```

Therefore, instead of storing a value of 1 in the variable called **i**, we've stored whatever is in the variable called **loopStart**. Whatever you type in the first text box used now as the starting value of the loop. For the second part, we have changed this:

i < 101

To this:

i < loopEnd

We are using the value stored inside of **loopEnd**. We are telling C# to keep looping if the value inside of the **i** variable is less than **loopEnd**. (Remember, because our Update Expression is **i++**, C# will keep adding 1 to the value of **i** each time round the loop. When **i** is no longer less than **loopEnd**, C# will stop looping). Run your program and type 1 in the first text box and 10 in the second text box. Click your button. You should find that the message box displays an answer of 45. Can you see a problem here? If you wanted to add up the numbers from 1 to 10, then the answer is wrong! It should be 55, and not 45. Can you see why 45 displayed in the message box, and not

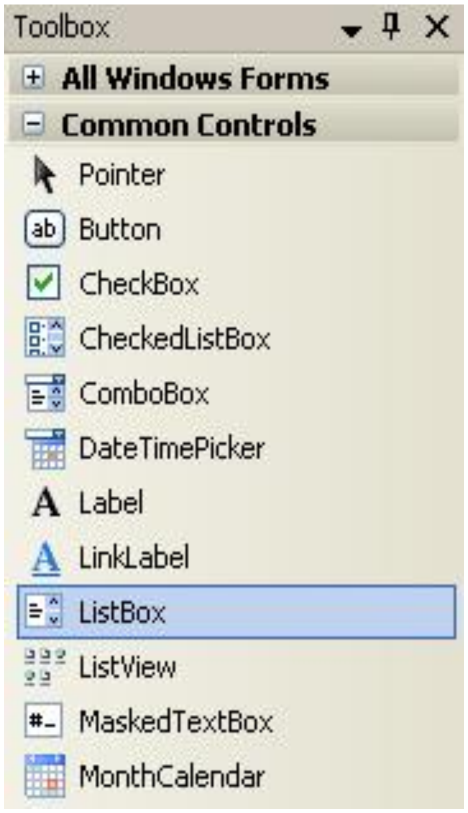
55? If you cannot, stop a moment and try to figure it out. (There is, of course, another problem. If you do not type anything at all in the text boxes, your program will crash! It does this because C# cannot convert the number from the text box and store it in the variable. After all, you cannot expect it to convert something that is not there! You will see how to solve this at the end of the chapter).

Times Table program

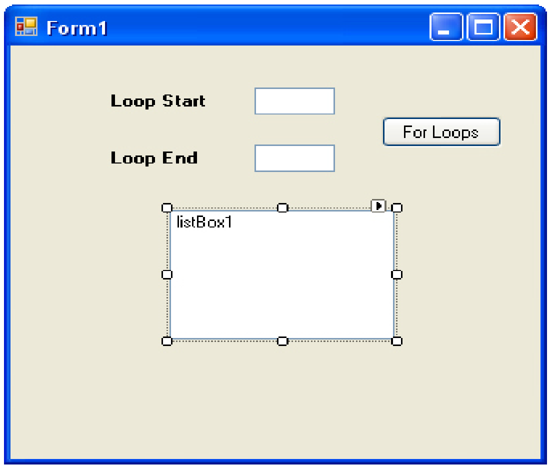
We can now write a little times table program. We will use the text boxes to ask users to input a start number and end number. We will use these to display the 10 times table. Therefore, if the user types a 1 into the first text box and a 5 into the second text box, we'll display this:

1 times 10 = 10
2 times 10 = 20
3 times 10 = 30
4 times 10 = 40
5 times 10 = 50

Instead of using a message box to display the results, we will use a List Box. A list box, you will not be surprised to hear, used to display lists of items. However, it is easier to show you what they do rather than explain. So use the Toolbox on the left of Visual Studio to add a list box to your form:



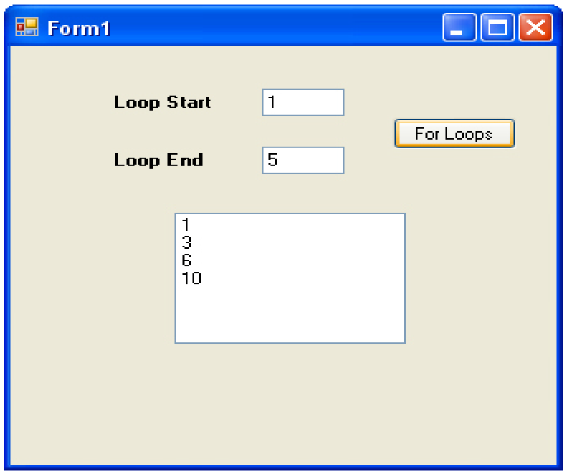
Resize your list box, and your form will now look something like ours below:



Double click your button to get at your code. Now add this line to your loop (the line to add is in bold):

```
for (int i = loopStart; i <= loopEnd; i++)
{
answer = answer + i;
listBox1.Items.Add( answer.ToString() );
}
```

Therefore, you start by typing the name of your list box. After a dot, you should see the IntelliSense list appear. Select **Items** from the list. **Items** is another property of list boxes. It refers to the items in your list. After the word Items, you type another dot. From the IntelliSense list, select the **Add** method. As its name suggests, the **Add** method adds items to your list box. Between the round brackets, you type what you want to add to the list of items. In our case, this was just the **answer**, converted **to** a **string**. You can delete the message box line, if you like, because you do not need it. However, run your program and enter 1 in the first text box and 5 in the second text box. Click your button and your form should look like this:



Loop Start: 1
Loop End: 5

```
1
3
6
10
```

The program is supposed to add up the number 1 to 5, or whatever numbers were typed in the text boxes. The list box is displaying one **answer** for every time round the loop. However, it is only displaying 4 items. If you solved the problem as to why 45 was displayed in the message box, and not 55 then you'll already know why there are only four items in the list box. If you did not, examine the first line of the for loop:

```
for (int i = loopStart; i < loopEnd; i++)
```

The problem is the second part of the loop code:

i < loopEnd

We are telling C# to go round and round while the value in **i** is **less than** the value in **loopEnd**. C# will stop looping when the values are equal. The value in **loopEnd** is 5 in our little program. Therefore, we are saying this to C#. "Keep looping while the value in i is less than 5. Stop looping if it's 5 or more". Cleary, we have used the wrong Conditional Operator. Instead of using the less than operator, we need … well, which one do we need?

Replace the " < " symbol with the correct one.