C# And Loops

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:

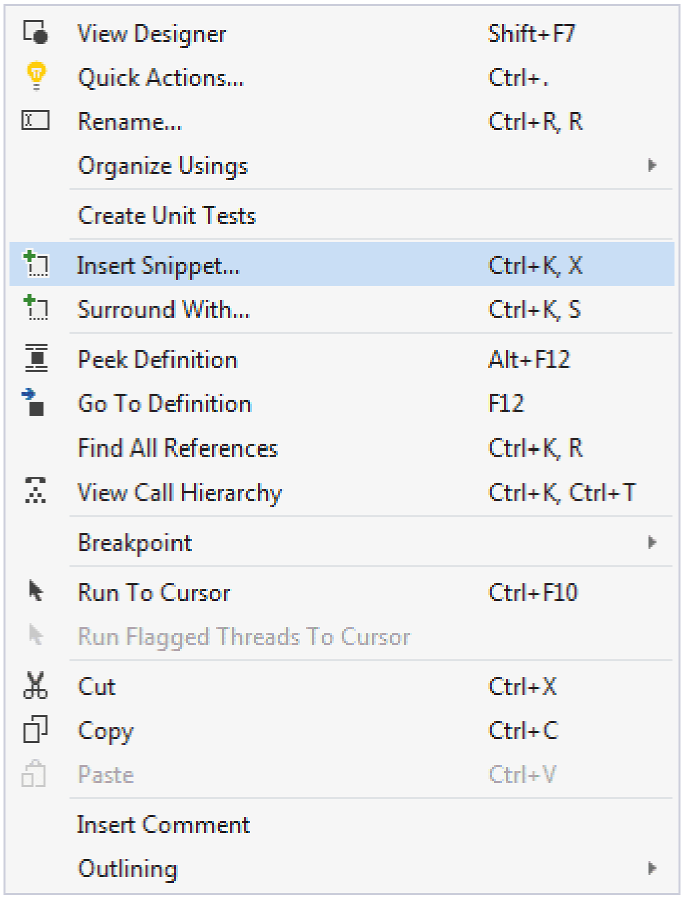
```
int answer;
answer = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;
```

And this would be OK if you only had 10 numbers. But suppose you had a thousand numbers, or ten thousand? You're certainly not going to want to type them all out! Instead, you use a loop to repeatedly add the numbers.

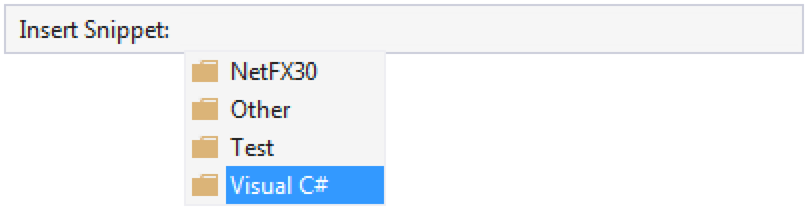
For Loops in C#

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

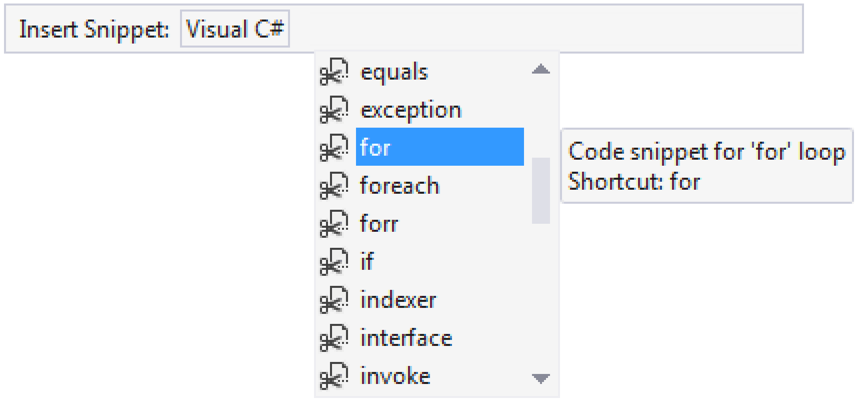
Start a new project by clicking **File > New Project** from the menu bars at the top of Visual Studio. Now add a button to the new form. Double click the button to get at the code. To quickly add a code stub for a loop, right click anywhere between the curly brackets of the button code. From the menu that appears, click on **Insert Snippet**:
(Click **Snippet > Insert Snippet** in Community 2017.)



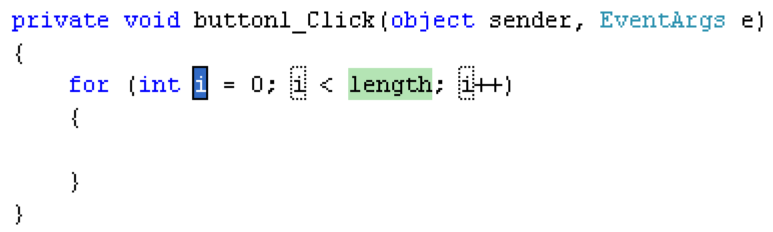
When you click on Insert Snippet, you'll see a list of items:



Double click the one for C# to see the list of snippets you can add:



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'll go through it. Here's the for loop without anything between the round brackets:

```
for ( )
{

}
```

So 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.

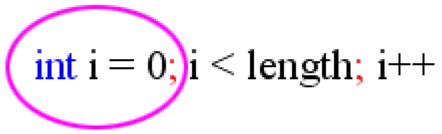
The default round-bracket code that C# inserts for you is this:

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

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

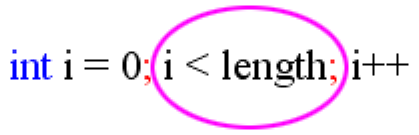
1. Which number do you want to start at?
2. How many times do you want to go round and round?
3. How do you want to update each time round the loop?

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



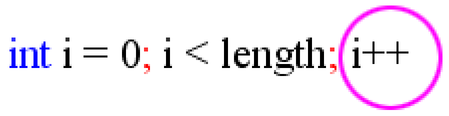
```
int i = 0; i < length; i++
```

And here's the second part:



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

And here's 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. So 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've just typed the i < 101. In the second example, we've set up a variable called length, and stored 101 in it. We're 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."

But you don't need to call the variable length. It's 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've 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. But C# doesn't 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

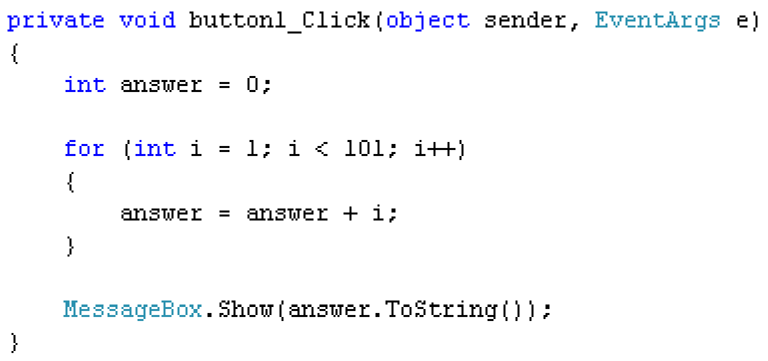
All you are doing is adding 1 to whatever is already inside of the variable name. Since you're in a loop, C# will keep adding 1 to the value of i each time round the loop. It only stops adding 1 to i when the end condition has been reached (i is no longer less than length).

So 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.

So your three parts are these:

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

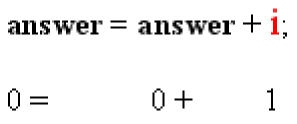
OK, time to put the theory into practice. Type the following for your button code:


```
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());
}
```

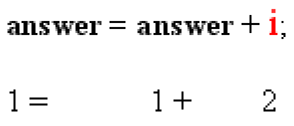
The actual code for the loop, the code that goes inside of the curly brackets, is this:

answer = answer + i;

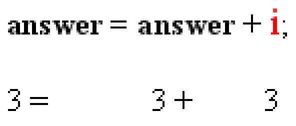
This is probably the trickiest part of loops - knowing what to put for your code! Just remember what you're trying to do: force C# to execute a piece of code a set number of times. We want to add up the numbers 1 to 100, and are using a variable called **answer** to store the answer to the addition. 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
```

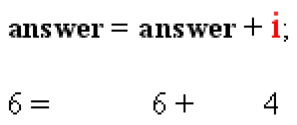
The second time round the loop, the figures are these:


```
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
```

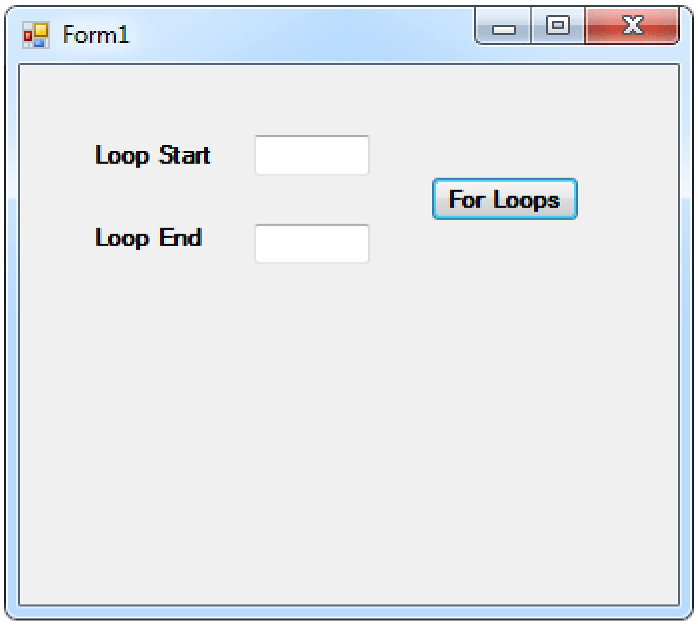
Notice how the value of **i** increases by one each time round the loop. If you first do the addition after the equals sign, the above will make more sense! (As an exercise, what is the value of answer the fifth time round the loop?)

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 from the previous page, 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'll do is to get the start value and end value from the text boxes. We'll then use these in our for loop.

So double click your button to get at the code (or press F7 on your keyboard). 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're 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

So 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 is now used as the starting value of the loop.

For the second part, we've changed this:

i < 101

to this:

i < loopEnd

We're using the value stored inside of **loopEnd**. We're 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 is displayed in the message box, and not 55? If you can't, stop a moment and try to figure it out.

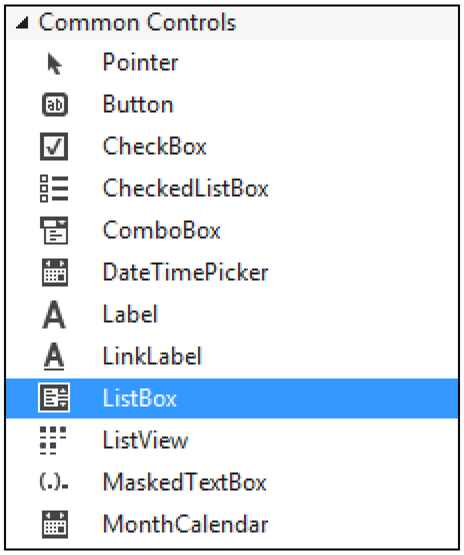
(There is, of course, another problem. If you don't type anything at all in the text boxes, your program will crash! It does this because C# can't convert the number from the text box and store it in the variable. After all, you can't expect it to convert something that's not there! You'll see how to solve this at the end of the chapter.)

A Times Table Program In C#

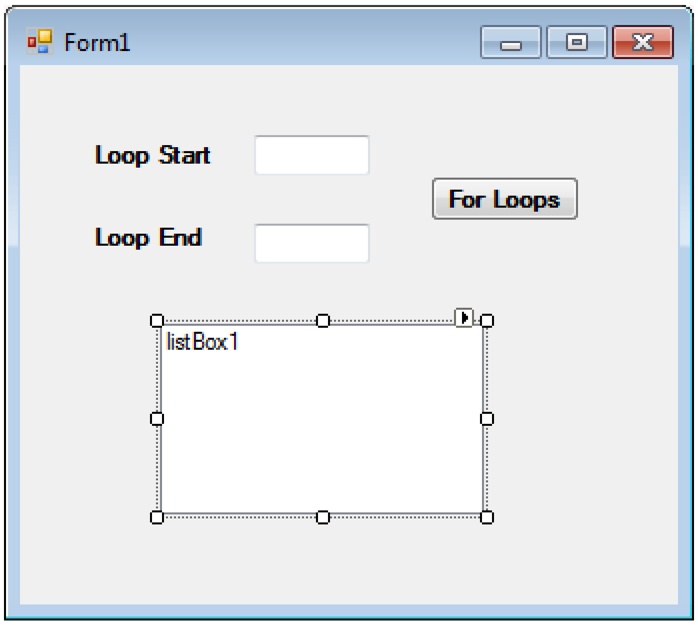
We can now write a little times table program. We'll use the text boxes to ask users to input a start number and end number. We'll use these to display the 10 times table. So if the user types a 1 into the first text box and a 5 into the second text box, we'll display this:

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'll use a List Box. A list box, you will not be surprised to hear, is used to display lists of items. But it's easier to show you what they do rather than explain. So use the Toolbox on the left of Visual C# to add a list box to your form:



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

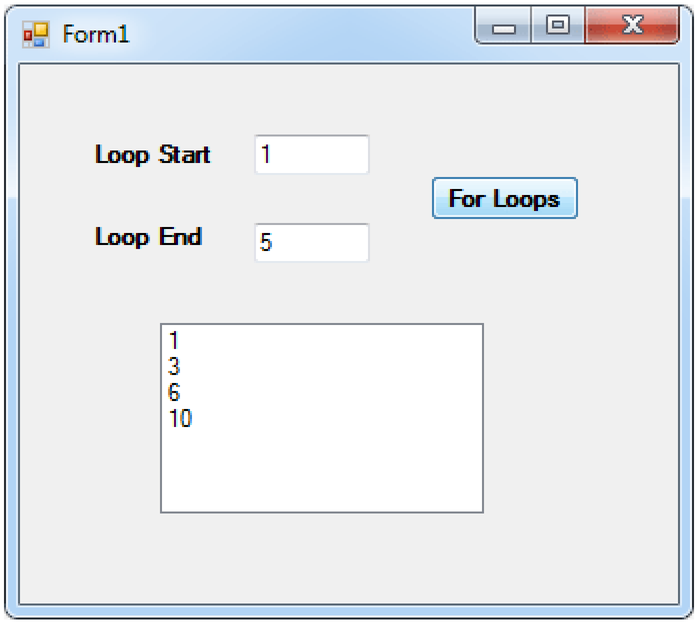


Double click your button to get at your code. Now add this line to your loop (the line to add is in blue bold below):

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

            answer = answer + i;

listBox1.Items.Add( answer.ToString() );

        }
```

So you start by typing the Name of your list box (**listBox1**, for us). 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 don't need it. But 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:



```
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's 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 didn't, 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're 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. So we're saying this to C#, "Keep looping while the value in i is less than 5. Stop looping if it's 5 or more."

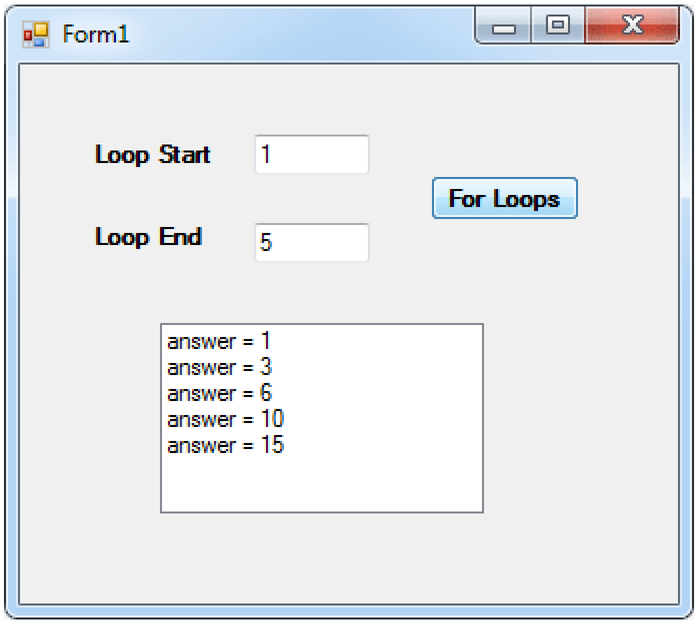
Cleary, we've used the wrong Conditional Operator. Instead of using the less than operator, we need … well, which one do we need? Replace the < symbol with the correct one.

Exercise G
For some extra points, can you think of another way to solve the problem? One where you can keep the less than symbol?

To add some more information in your list box, change your line of code to this:

```
listBox1.Items.Add( "answer = " + answer.ToString( ) );
```

The thing to add is the text in red above. We've typed some direct text **"answer="** and followed this with the concatenation symbol (**+**). C# will then join the two together, and display the result in your list box. Run your program again, and the list box will be this:

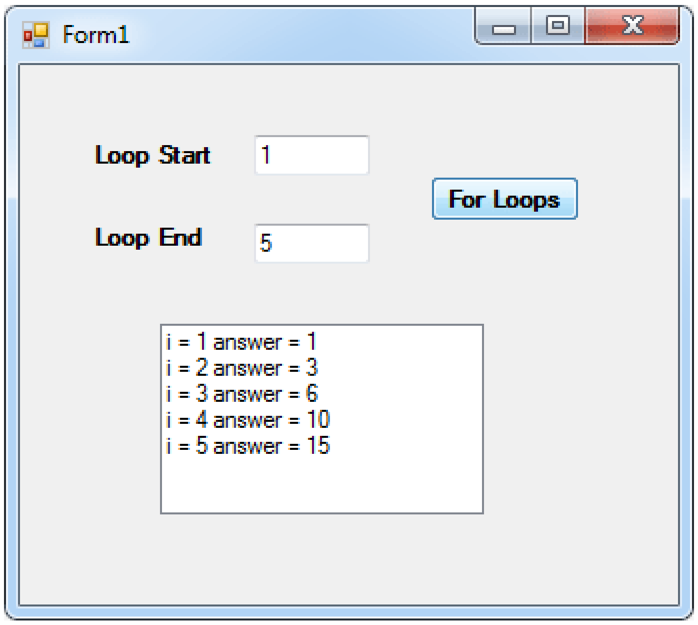


```
answer = 1
answer = 3
answer = 6
answer = 10
answer = 15
```

To make it even clearer, add some more text to your list box. Try this:

listBox1.Items.Add("i = " + i + " answer = " + answer.ToString( ));

Again, the text to add is in blue. Run your program, and your list box will look like this:



```
i = 1 answer = 1
i = 2 answer = 3
i = 3 answer = 6
i = 4 answer = 10
i = 5 answer = 15
```

We've now added the value of the variable called i. This makes it clear how the value of i changes each time round.

The Times Table Program

We now have all the ingredients to write the Times Table program.

So return to your coding window. What we're going to do, remember, is to use a for loop to calculate and display the 10 times table. We need another variable, though, to hold the 10. So add this to your variables:

int multiplyBy = 10;

The only other thing we need to do is to change the code between the curly brackets of the for loop. At the moment, we have this:

```
answer = answer + i;
listBox1.Items.Add( "i = " + i + " answer = " +
answer.ToString() );
```

Delete these two lines and replace them with these two:

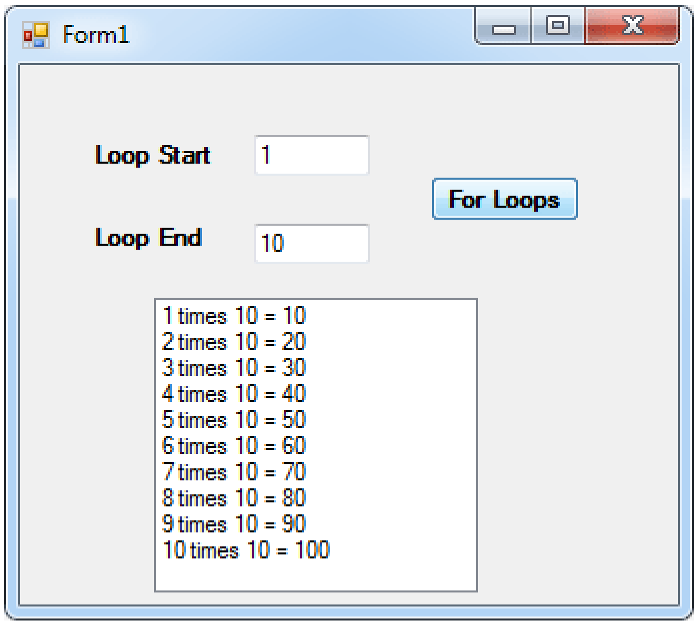
```
answer = multiplyBy * i;

listBox1.Items.Add(i + " times " + multiplyBy + " = " +
answer.ToString());
```

The list box line is a bit messy, but examine the part between the round brackets:

```
i + " times " + multiplyBy + " = " + answer.ToString()
```

It's just a combination of variable names and direct text. The first line of the code, though, is a simple multiplication. We multiply whatever is inside of the variable called multiplyBy (which is 10) by whatever is inside of the variable called i. C# is adding 1 to the value of i each time round the loop, so the answer gets updated and then displayed in the list box.

Run your program. Enter 1 in the first text box and 10 in the second text box. Click you button to see the ten times table:

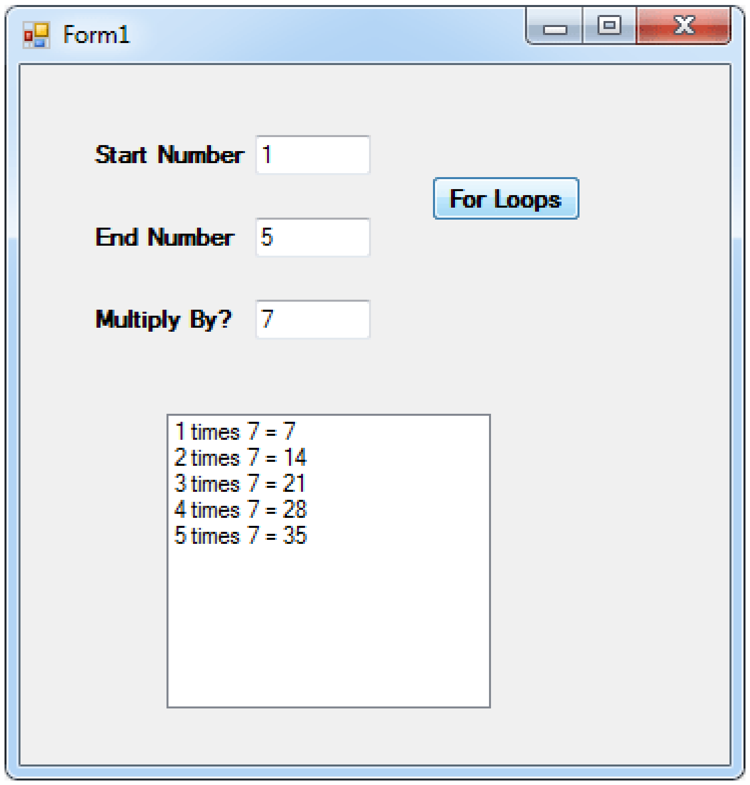


```
1 times 10 = 10
2 times 10 = 20
3 times 10 = 30
4 times 10 = 40
5 times 10 = 50
6 times 10 = 60
7 times 10 = 70
8 times 10 = 80
9 times 10 = 90
10 times 10 = 100
```

So with just a few lines of code, and the help of a **for** loop, we've created a program that does a lot of work. Think how much more difficult this type of program would be without looping.

Exercise H

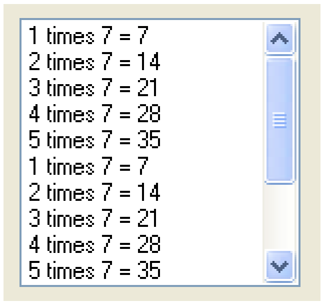
At the moment, we're multiplying by 10 and, therefore, displaying the 10 times table. Add another text box to your form. Add a label that asks users which times table they want. Use this value in your code to display the times table for them. So if your user types a 7 into your new text box, clicking the button would display the 7 times table in the list box. Here's what your program should look like when you're finished (we've change the first two labels, as well):



```
1
5
7
```

```
1 times 7 = 7
2 times 7 = 14
3 times 7 = 21
4 times 7 = 28
5 times 7 = 35
```

When you complete this exercise, click your button a few times. You'll notice (as long as you have numbers in the text boxes) that the list box doesn't clear itself. You'll have this in your list box:



```
1 times 7 = 7
2 times 7 = 14
3 times 7 = 21
4 times 7 = 28
5 times 7 = 35
1 times 7 = 7
2 times 7 = 14
3 times 7 = 21
4 times 7 = 28
5 times 7 = 35
```

So the new information is simply added to the end. To solve this, add the following line anywhere before your for loop, but inside of your button code.

listBox1.Items.**Clear()**;

So instead of using the Add() method, you use the Clear() method. This will clear out all the items in a list box. But can you see why the line of code would be no good inside of the loop? Or after the loop?

But that's enough of **for** loop. We'll now briefly explore two other types of loops: do loops and while loops.

C# Do Loops And While Loops

As well as using a **for** loop to repeatedly execute some code, you can use a **Do** loop or a **While** loop. We'll start with the Do Loop.

C# Do Loops

Whichever loop you use, the idea is still the same: go round and round and execute the same code until an end condition is met. The difference with the Do and While loops is in the structure. Here's what the Do loop looks like:

```
do
{

} while (true);
```

Notice where the semi-colon is, in the code above. It comes right at the end, after the round brackets. But you start with the word **do**, followed by a pair of curly brackets. After the curly brackets, you type the word **while**. After while, and in between some round brackets, you type your end condition. C# will loop round and round until the end condition between the round brackets is met. Only then will it bail out. Here's an example, using our times table program:

```
do
{

    answer = multiplyBy * i;
    listBox1.Items.Add(answer.ToString());
    i++;

} while (i <= loopEnd);
```

So this time, we've used a Do Loop instead of a For Loop. The loop will go round and round while the value in the variable called i is less than or equal to the value in the variable called loopEnd. The other thing to notice here is we have to increment (add one to) the value in i ourselves (i++). We do this each time round the loop. If we didn't increment the value in i then it would always be less than loopEnd. We'd have then created an infinite loop, and the program would crash. But we're really saying this:

"Keep Doing the code in curly brackets while i is less than or equal to loopEnd."

C# While Loops

While loops are very similar in structure to Do loops. Here's what they look like:

```
while (true)
{

}
```

And here's the times table code again:

```
while (i <= loopEnd)
{

    answer = multiplyBy * i;
    listBox1.Items.Add(answer.ToString());
    i++;

}
```

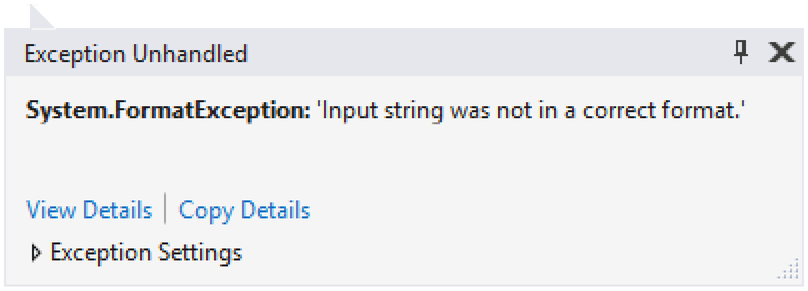
While loops are easier to use than Do loops. If you look at the code above, you can see that the while part is at the start, instead of at the end like a Do Loop. The code you need to execute repeatedly still goes between curly brackets. And you still need a way for the loop to end (i++).

The difference between the two loops is that the code in a Do loop will get executed at least once, because the while part is at the end. With the while part at the beginning, your end condition in round brackets can already be true (i might be more than loopEnd). In which case, C# will bail out immediately, and the code in curly brackets won't get executed at all.

Deciding which loop to use can be quite tricky. Don't worry if you haven't fully understood how to use loops. You'll get lots more practice as you work your way through this book. But loops are difficult to get the hang of, and you shouldn't consider yourself a failure if you haven't yet mastered them. For now, we'll leave this complex subject, and end the section with a problem, and a solution.

Checking For Blank Textboxes In C#

There's a problem with the text boxes on your times table program. If you don't type anything at all in the text boxes, your program will crash! Try it out. Start your program and leave the text boxes blank. Now click your button. You should get a strange and unhelpful error message:



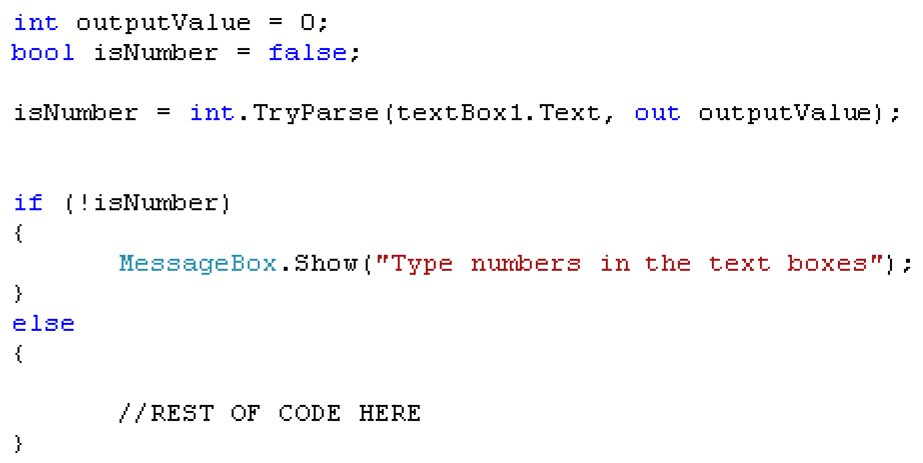
```
System.FormatException: 'Input string was not in a correct format.'
```

C# is highlighting the offending line in yellow. It does this because it can't convert the numbers from the text box and store them in the variables. After all, you can't expect it to convert something that's not there! To remedy this, you can use a method called **TryParse**.

To convert the numbers from the text boxes to integers, you've been doing this:

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

So you've Parsed the number in the text box, and turned it into an int. But this won't check for blank text boxes, and it won't check to see if somebody typed, say, the word three instead of the number 3. What you need to do is to **Try** and **Parse** the data in the text box. So you ask C# if it can be converted into a number. If it can't, you display an error message for your users. Here's some code that tries to parse the data from the first text box. It's a bit complex, so we'll go through it.



```
int outputValue = 0;
bool isNumber = false;

isNumber = int.TryParse(textBox1.Text, out outputValue);


if (!isNumber)
{
       MessageBox.Show("Type numbers in the text boxes");
}
else
{

       //REST OF CODE HERE
}
```

The first two lines set up some variables, an integer and a Boolean. The **outputValue** is needed for **TryParse**. You are trying to output a Boolean value (true or false) AND the string of text:

```
isNumber = int.TryParse(textBox1.Text, out outputValue);
```

So **isNumber** will be either **true** or **false**, depending on whether or not C# can convert the text box data into an integer. If you were trying to parse a double variable your code would be this, instead:

```
double outputValue = 0;
bool isNumber = false;

isNumber = double.TryParse(textBox1.Text, out
outputValue);
```

The output value that you need is now a **double** (You're checking to see if C# can convert to a double value). The value of **isNumber** will still be either **true** or **false** (can it be converted or not).

After using **TryParse**, you then need to check that true or false value:

```
if (!isNumber)
{

     MessageBox.Show("Type numbers in the text boxes");
}
else
{

    //REST OF CODE HERE

}
```

If you remember the lesson on Conditional Operators, you'll know that this line:

if (**!**isNumber)

reads this:

If NOT true

If you prefer, you can write the line like this:

if (isNumber == false)

The line now reads:

"If isNumber has a value of false"

If isNumber is false, then you display an error for your users. If it's true, then it means that data from the text box can be converted properly. In which case, the rest of the code goes between the curly brackets of else.

If all that is a bit too complex then don't worry about it - you'll get there! In the next section, you'll be doing something far easier than loops and Conditional Logic: we'll show you how to add menus to your programs.

But when you are first starting out, these are the two biggest hurdles to overcome: loops and Conditional Logic. When you understand these two difficult subjects then you are well on your way to becoming a program!